2023

A Novel Regularized Orthonormalized Partial Least Squares Model for Multi-view Learning

Ce Bian

A Novel Regularized Orthonormalized Partial Least Squares Model for Multi-view Learning

by

CE BIAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2023

To my parents Hong Gao and Lisheng Bian,

and all of my friends,

who give me a happy life.

Acknowledgement

First of all, I would like to thank my supervising professors Dr. Li Wang and Dr. Ren-Cang Li for constantly advising and encouraging me. Their invaluable advice, deep expertise in various computer science sub-domains and huge breadth in data science as a whole has proved to be extremely helpful during the entire course of my graduate studies. I wish to thank my thesis committee members Dr. Shan Sun-Mitchell and Dr. Dengdeng Yu for their interest in my research and for taking time to be in my thesis committee.

I would also like to extend my appreciation to the mathematics department at UT Arlington for providing financial support for my studies and a stimulating working environment for continuing my research on many topics I have been interested in. I am especially grateful to Dr. Mei Yang for her interest in my research and for the helpful discussions and invaluable comments. I wish also to thank my cousin Dr. Yajing Bian for taking the time to critically evaluate this manuscript and my research papers.

I am grateful to all the professors who taught me during the years I spent as a graduate student.

Finally, I would like to express my deep gratitude to my parents who have encouraged me and also sponsored my graduate studies.

August 10th, 2023

Abstract

A Novel Regularized Orthonormalized Partial Least Squares Model for Multi-view Learning

Ce Bian, Ph.D.

The University of Texas at Arlington, 2023

Supervising Professors: Dr. Li Wang, and Dr. Ren-Cang Li

Over the past few years, the size of data dimensions or features has been increasing in various fields of science and engineering, owing to the rapid pace of data collection and the development of more advanced storage methods. To handle high-dimensional data, dimensionality reduction is essential before performing classification or regression tasks to eliminate noisy features. There are several numerical methods available for reducing data dimensionality, such as Canonical Correlation Analysis (CCA), Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA). While these methods offer valuable approaches to data dimensionality reduction, they do come with certain limitations. CCA, for instance, primarily focuses on finding correlations between two sets of variables, which might not fully capture the complexities of intricate relationships within multidimensional data. PCA, while excellent at preserving variance, can struggle to emphasize class separability when applied to classification tasks.

Acknowledging these limitations, this thesis introduces an innovative supervised dimensionality reduction algorithm that tackles the reduction of data dimen-

sionality and the classification of the data concurrently, simultaneously. Unlike conventional methods, this algorithm embarks on the dual task of revealing the projection matrix for dimension reduction alongside identifying the classifier hyperplane for data classification. The result is a model that excels in both accuracy and efficiency, enabled by its simultaneous learning of low-dimensional representation and classification models.

What distinguishes this proposed model is its versatility. It accommodates not only the dimensionality reduction and classification of single-view data but also extends its prowess to multi-view data. Through numerical simulations, the effectiveness and computational efficiency of the proposed model are showcased when contrasted against state-of-the-art methods in dimensionality reduction and classification.

A noteworthy feature of this novel approach is its capacity to generate two classifiers in tandem. This unique attribute widens its applicability across diverse classification experiments encompassing a variety of data types. In effect, the method's dual-classifier capability amplifies its utility and establishes it as a versatile choice for tackling complex classification challenges.

Keywords: Dimensionality Reduction, Multi-view Learning, Subspace Learning, Classification

TABLE OF CONTENTS

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Multi-view learning is an increasingly active research area in machine learning that involves the use of multiple data schemas or views [46]. These views may be complementary or redundant, and integrating them can enhance predictive performance [70]. There are several established multi-view learning techniques, which have been extensively employed in diverse fields such as genetics, imaging, chemometrics, and finance [68, 90]. Recently, orthonormalized partial least squares (OPLS) [39, 79] has emerged as a promising extension of partial least squares (PLS) [6, 55]. OPLS improves the interpretability of PLS models by orthogonalizing data variations that are unrelated to the response variable, thereby facilitates the identification of relevant features and enhances generalization performance [25, 39, 79].

This thesis introduces a novel multi-view learning approach, Multi-view Regularized Orthonormalized Partial Least Squares (Multi-view R-OPLS), which builds upon the OPLS methodology. Multi-view R-OPLS is designed to handle multiple views of data and utilizes complementary information from these views to enhance predictive performance. To achieve this, Multi-view R-OPLS combines support vector machine (SVM) [16, 50, 54, 88] to maximize the covariance between the view and response variables, while orthogonalizes the variation unrelated to the response.

We assess the effectiveness of Multi-view R-OPLS through extensive experiments on several benchmark data sets. We compare its performance with that of various state-of-the-art dimensionality reduction techniques, such as Canonical Correlation Analysis (CCA) [37], Linear Discriminant Analysis (LDA) [72], Principal

Component Analysis (PCA) [31, 68, 74], Multi-view Canonical Correlation Analysis (MCCA) [56, 76], Multi-view Linear Discriminant Analysis (MLDA) [15, 72], Multi-view Principal Component Analysis (MPCA) [15, 31], and Generalized Multi-view Analysis (GMA) [68]. Our experimental findings demonstrate that Multi-view R-OPLS outperforms these techniques in terms of classification accuracy, particularly in situations involving complementary or noisy views.

This thesis is structured as follows: in Chapter 2, we present the fundamental concepts and background knowledge necessary for understanding the proposed method. In Chapter 3, we provide a comprehensive overview of the Multi-view R-OPLS framework, including the model formulation, mathematical derivations, and optimization algorithms. We also discuss the experimental setup for single-view binary classification data, followed by experimental results on various benchmark datasets. Chapter 4 presents the experimental setup for single-view multi-classification data, followed by experimental results on several benchmark datasets. In Chapter 5, we set up experiments for multi-view multi-classification data and present experimental results on various benchmark datasets. Finally, we conclude the thesis with Chapter 6.

CHAPTER 2

Related Works

2.1 Least Squares

In data science research, one of the most common and critical tasks is to find the optimal solution of a linear or nonlinear regression equation that expresses the relationship between dependent and independent variables [10]. This process is essential for quantitative forecasting and understanding trends in results, which is crucial for making informed decisions based on data analysis [10, 62]. There are many methods available for regression analysis, including logistic regression [10], random forest regression [4, 14, 17], and least squares [50, 62] method, to name a few. In the real world, predicting values can be challenging, as there is often a deviation between the predicted value and the actual value [10]. This difference between the predicted and actual values is known as an error or residual. When we attempt to fit a line of best fit through a data set, some residuals will be positive, while others will be negative [25, 80]. In other words, some predicted values will be greater than the actual values, while others will be less than the actual values [10]. In order to minimize these errors, we need to sum the residuals. However, cancellation errors may occur when subtracting two numbers with very similar values, as the common leading digits are canceled out. This can result in the sum of residuals being zero, leading to a potential cancellation error [24, 62].

Therefore, the least squares method is widely used to determine the line of best fit for a given data set, which provides a visual representation of the relationship between data points. This mathematical method has become one of the most popular

and efficient approaches to regression analysis [10]. The fundamental principle of the least squares method is to derive an optimal linear approximation that minimizes the sum of squared errors, also referred to as squared residuals. These errors arise from the discrepancies between the actual and predicted values [10]. By minimizing the squared error, the least squares method ensures that the sum of the squared residuals is the smallest possible value, thereby minimizing the impact of outliers and reducing the chances of cancellation errors [60]. By doing so, the least squares method aids in determining the optimal regression curve, which accurately represents the underlying relationship between the dependent and independent variables [10, 60]. This regression curve serves as the most fitting approximation of their association. This method has several advantages, including its uniqueness, convenience, and good analytical properties [62]. Despite its strengths, the least squares method also has its limitations. It is significantly impacted by the presence of outliers, which can lead to biased and inaccurate results. Therefore, it is essential to identify and address outliers before applying the least squares method to regression analysis [62].

Least squares classification method adapts linear regression for classification, and uses least squares error as the loss function. let $\{(x_i,\ y_i)\}_{i=1}^n$ be a set of n training samples with c class labels, where $x_i \in \mathbb{R}^d$ and the label information $y_i \in \{1, \ldots, c\}$. Denote the input data matrix $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$ and the indicator matrix $Y \in \{0,\ 1\}^{c\times n}$ by using one-hot representation of class labels. The goal of least squares classification is to learn a classifier that can predict the correct label to test samples, by solving

$$\min_{W} \quad f_{ls}(W) := \|Y - (W^T X + \boldsymbol{b}\mathbf{1}_n^T)\|_F^2, \tag{2.1}$$

where the coefficient matrix $W \in \mathbb{R}^{d\times c}, \boldsymbol{b} = [b_1, \ldots, b_c]^T \in \mathbb{R}^c$ is a bias vector, $\mathbf{1}_n$ is the column vector with all 1s, and $\|\cdot\|_F$ is the Frobenius norm.

One-hot representation [27, 44] is a crucial technique in machine learning that converts discrete numerical labels into a format that can be easily processed by machine learning algorithms [22, 44]. The process involves transforming the labels into a $\{0,\ 1\}$ matrix, where each row corresponds to a class label, and each column coresponds to a data point. In this matrix, only the position corresponding to the original label is set to 1, and all other positions are set to 0. As a result, each column in the matrix contains only one 1 and the rest are 0s [22].

In the context of the least squares method, the loss function is defined as the sum of squared errors between the predicted value and the actual value. When using discrete numerical labels, the loss between different label categories is unequal, leading to incorrect model judgment [22, 44]. For instance, the loss when predicting label 3 as label 1 is greater than the loss when predicting label 3 as label 2, which is inconsistent with the actual scenario. This is the reason why one-hot representation comes into play. By using one-hot representation, the distances between different classes are guaranteed to be the same, making it more appropriate for multi-class classification [22, 44]. This ensures that the loss function treats all classes equally and provides a more accurate assessment of the model's performance.

Moreover, one-hot representation also helps to avoid the issue of ordinality, where the machine learning algorithm may treat the labels as being in some order [44]. This can result in poor performance of the model as it may fail to distinguish between the different classes correctly [44]. One-hot representation resolves this issue by treating each class as independent and non-ordinal [44].

Once the indicator matrix has been obtained, the next step is to centralize the data. Centralization is a statistical technique that involves transforming the data to have a mean of zero [23]. This is done to remove any systematic differences between the data and to make it easier to compare different variables in the analysis.

Centralizing the data involves subtracting the mean from each observation in the data set [7, 22]. This results in a new data set where the mean of each variable is equal to zero. Centralization is a necessary step in regression analysis because it helps to avoid issues such as multicollinearity, which can occur when there is a high correlation between the independent variables in the model [41].

In addition to addressing multi-collinearity, centralization has several other benefits. One of the main benefits is that it simplifies the interpretation of the regression coefficients [13, 80]. When the data is centralized, the intercept term in the regression equation represents the predicted value of the dependent variable when all the independent variables are equal to zero [10]. This can be a more meaningful interpretation of the intercept, as it represents a more realistic starting point for the analysis [23].

Let $\hat{X} = XH \in \mathbb{R}^{d\times n}$, $\hat{Y} = YH \in \mathbb{R}^{c\times n}$ be the centered of X and Y, where $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T \in \mathbb{R}^{n\times n}$ is the centering matrix, I_n is the identity matrix of size n, and $\mathbf{1}_n$ is the n-dimensional column vector of all ones. The problem (2.1) is reformulated as a regression problem in the mean square error sense [6, 63, 79]:

$$\min_{W} \quad f_{ls}(W) := \|\hat{Y} - W^T\hat{X}\|_F^2. \tag{2.2}$$

Assuming that the matrix $\hat{X}\hat{X}^T$ is positive definite (this will be resolved later by introducing regularization) [79], the first-order optimality condition with respect to W is

$$\frac{\partial f_{ls}(W)}{\partial W} = -2\hat{X}(\hat{Y} - W^T\hat{X})^T = 0, \tag{2.3}$$

$$\Rightarrow \hat{X}\hat{Y}^T = \hat{X}\hat{X}^T W, \tag{2.4}$$

$$\Rightarrow W = (\hat{X}\hat{X}^T)^{-1}\hat{X}\hat{Y}^T. \tag{2.5}$$

However, in the real world, $(\hat{X}\hat{X}^T)^{-1}$ might not exist because $\hat{X}\hat{X}^T$ might be a singular matrix. For any matrix $\hat{X} \in \mathbb{R}^{d\times n}$,

$$\text{rank}(\hat{X}) = \text{rank}(\hat{X}^T) = \text{rank}(\hat{X}\hat{X}^T) = \text{rank}(\hat{X}^T\hat{X}) \leq \min\{d,\ n\}.$$

Therefore, to avoid the problem of $\hat{X}\hat{X}^T$ being a singular matrix, we introduce the regularization such as $(\hat{X}\hat{X}^T + \epsilon I_d)$, where ϵ is a small positive number.

Afterwards, the optimal solution of W is approximated as

$$W = (\hat{X}\hat{X}^T + \epsilon I_d)^{-1}\hat{X}\hat{Y}^T. \tag{2.6}$$

2.2 Orthonormalized Partial Least Squares

Orthonormalized partial least squares (OPLS) [79] is a relatively new and advanced technique for modeling multi-class data that has been gaining attention in recent years [39, 42]. Unlike traditional least squares, OPLS is a supervised multi-class data projection method that associates a set of predictors with one or more responses. The purpose of this technique is to reduce the model complexity by reducing the number of latent variables, while identifying, analyzing and studying the orthogonal variation that exists in the data [19, 25].

OPLS has many benefits over traditional least squares when it comes to multi-view classification [42]. While both methods have similar predictive power when fitted to single feature data, OPLS offers a distinct advantage in terms of model interpretability [79, 84]. Specifically, OPLS is able to split the explained variance into predictive and orthogonal compartments, which greatly simplifies the model interpretation process [19, 42]. This feature makes OPLS a valuable tool for data scientists who need to explain the relationship between variables in a clear and concise manner.

In addition to its interpretability advantages, OPLS has also been shown to perform better than traditional least squares in certain scenarios [79]. For example, in cases where there are many correlated predictors, OPLS can identify and eliminate the redundant variables to improve model performance [42]. Similarly, in situations where the predictors and responses are highly correlated, OPLS can identify and remove the correlation to improve model accuracy [19, 42].

OPLS model aims to learn a projection matrix $P \in \mathbb{R}^{d\times k}$ with $k < d$ to transform input data from a d-dimensional space $\mathbb{R}^d$ to a k-dimensional space $\mathbb{R}^k$ by solving [79]

$$\begin{aligned} \min_{P,W} \quad & f_{opls}(P,W) := \|\hat{Y} - W^T P^T \hat{X}\|_F^2 \\ \text{s.t.} \quad & P^T P = I_k, \end{aligned} \tag{2.7}$$

where the coefficient matrix $W \in \mathbb{R}^{k\times c}$.

The first-order optimality condition of (2.7) with respect to W is

$$W = (P^T \hat{X}\hat{X}^T P)^{-1} P^T \hat{X}\hat{Y}^T. \tag{2.8}$$

As we mentioned in previous section, we also make $\hat{X}\hat{X}^T$ as a full rank matrix to avoid the problem of $\hat{X}\hat{X}^T$ being a singular matrix. Then the optimal solution of W is approximated to

$$W = (P^T(\hat{X}\hat{X}^T + \epsilon I_d)P)^{-1} P^T \hat{X}\hat{Y}^T. \tag{2.9}$$

Now, substituting (2.9) into (2.7), we can have a reformulated objective function

$$f_{opls}(P) := \|\hat{Y} - [(P^T(\hat{X}\hat{X}^T + \epsilon I_d)P)^{-1} P^T \hat{X}\hat{Y}^T]^T P^T \hat{X}\|_F^2. \tag{2.10}$$

Therefore, problem (2.7) can be reformulated as follows

$$\begin{aligned} \min_{P} \quad & \|\hat{Y}\|_F^2 - \mathrm{tr}[(P^T(\hat{X}\hat{X}^T + \epsilon I_d)P)^{-1} P^T \hat{X}\hat{Y}^T \hat{Y}\hat{X}^T P] \\ \text{s.t.} \quad & P^T P = I_k. \end{aligned} \tag{2.11}$$

Here, we define $U = P^T(\hat{X}\hat{X}^T + \epsilon I_d)P$, with U being a symmetric positive definite matrix. Utilizing Cholesky decomposition, we have $U = L^T L$, where L is a lower triangular matrix. Furthermore, $U^{-1} = L^{-1}L^{-T}$. To facilitate the optimization process, we introduce $\hat{P} = PL^{-1}$. Consequently, $\hat{P}^T = L^{-T}P^T$.

According to the trace property, the objective function of problem (2.11) is

$$\begin{aligned}
&-\text{tr}[(P^T(\hat{X}\hat{X}^T + \epsilon I_d)P)^{-1}P^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T P] \\
&= -\text{tr}(U^{-1}P^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T P) \\
&= -\text{tr}(L^{-1}L^{-T}P^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T P) \\
&= -\text{tr}(L^{-T}P^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T PL^{-1}) \\
&= -\text{tr}(\hat{P}^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T\hat{P}).
\end{aligned}$$

Also,

$$\begin{aligned}
&\hat{P}^T(\hat{X}\hat{X}^T + \epsilon I_d)\hat{P} = (PL^{-1})^T(\hat{X}\hat{X}^T + \epsilon I_d)PL^{-1} \\
&= L^{-T}P^T(\hat{X}\hat{X}^T + \epsilon I_d)PL^{-1} = L^{-T}UL^{-1} = L^{-T}L^T LL^{-1} = I_k.
\end{aligned}$$

Therefore, the OPLS model of problem (2.11) is equivalent to

$$\begin{aligned}
\max_{\hat{P}} \quad & \text{tr}(\hat{P}^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T\hat{P}) \\
\text{s.t.} \quad & \hat{P}^T(\hat{X}\hat{X}^T + \epsilon I_d)\hat{P} = I_k.
\end{aligned} \tag{2.12}$$

The optimization problem (2.12) can be considered as a generalized eigenvalue problem [26] on the matrix $\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T$ with regularization term $\hat{X}\hat{X}^T + \epsilon I_d$. The optimal solution for $\hat{P}$ corresponds to the eigenvectors of $\hat{X}\hat{Y}^T\hat{Y}\hat{X}^T$ with $\hat{X}\hat{X}^T + \epsilon I_d$ that correspond to the top k eigenvalues [22]. Here, ϵ is a small positive number, typically set to 10^{-8}. In the subsequent algorithm section, we will delve into the details of the generalized eigenvalue problem.

2.3 Support Vector Machines

Before diving into the proposed model, we briefly introduce SVM [88] for classification. SVM is also one of the most popular and efficient supervised machine learning classification model [16]. Since SVM can produce significant accuracy with less computation, it is highly preferred by many researchers [13, 88].

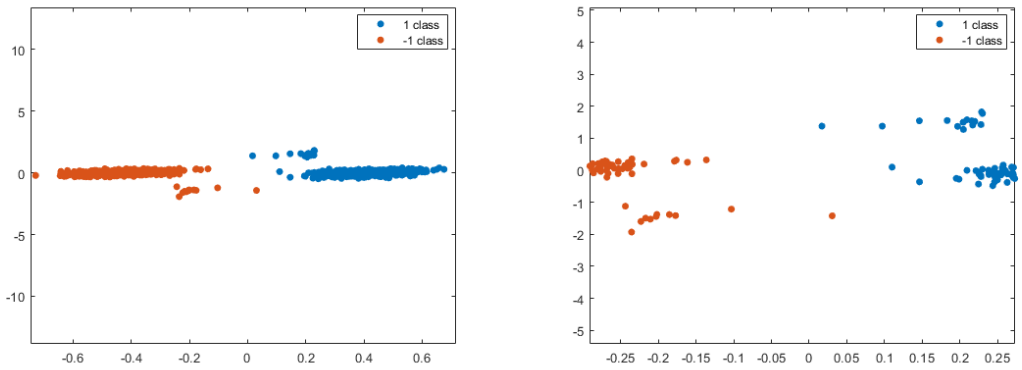


Figure 2.1: Binary Class Data Sets

Given data set $\{(x_i,\ y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1,\ 1\}$, the goal of binary classification is to learn a function $f:\ \mathbb{R} \to \{-1,\ 1\}$. In Figure 2.1, there are two types of data, blue and orange. SVM is a good choice when we want to use machine learning to classify two classes of data and predict the class of unknown data. A hyperplane is a decision boundary that helps classify data points and enables data points that fall on either side of the hyperplane to be assigned to different classes [54, 88]. To separate these two classes of data points, many possible hyperplanes can be chosen as shown in Figure 2.2. Furthermore, the number of features of the data points determines the dimension of the hyperplane [50]. If the input data is two-dimensional, then the hyperplane is a line. If the input data is three-dimensional, then the hyperplane becomes a two-dimensional plane. It's hard to imagine when the

number of features exceeds three. That is, if the data has more than three features, the visualization is not easy to achieve [50, 88].

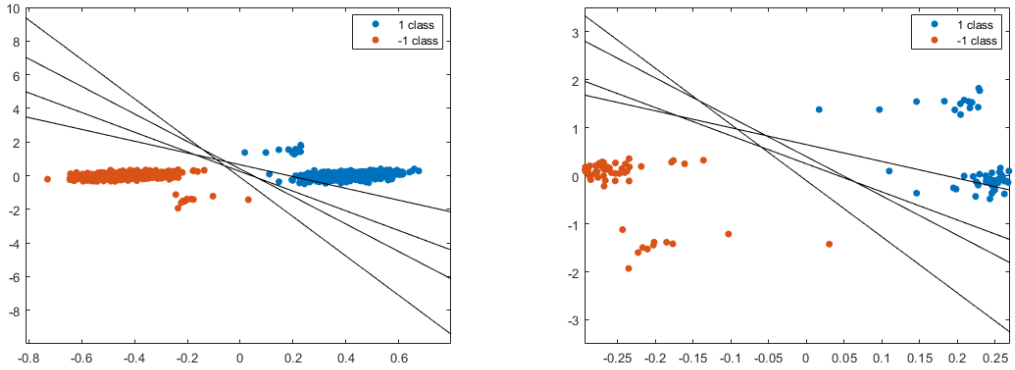


Figure 2.2: Hyperplanes in SVM

Support vectors are data points that are closer to the hyperplane and affect the position and orientation of the hyperplane [88]. Using these support vectors, we maximize the margins of the classifier. Deleting the support vector changes the position of the hyperplane. These are the points that helped us build the SVM. In SVM, we use margin, i.e. the largest distance between two classes of data points, to describe the distance between hyperplane and support vectors [50]. Maximizing the margin distance provides some reinforcement so that future data points can be classified more accurately [16]. The goal of SVM is to find the optimal hyperplane with the largest margin as shown in Figure 2.3. Before introducing margin, we need to know how to calculate the distance from a point in space to a plane. SVM want to find the linear coefficient of a classifier v such that $v^T x_{(\text{class1})} \geq 1$ and $v^T x_{(\text{class2})} \leq -1$, when x_i is in class $y_i = 1$, and x_i is in class $y_i = -1$. Then one can have $y_i(v^T x_i) \geq 1$, also $y_i(v^T x_i) - 1 \geq 0$, margin $\geq \frac{v^T}{\|v\|_2}(x_{\text{class1}} - x_{\text{class2}}) = \frac{2}{\|v\|_2}$ [60, 88]. It is equivalent to solving the problem $\min \frac{1}{2}\|v\|_2^2$, for mathematically convenient [88].

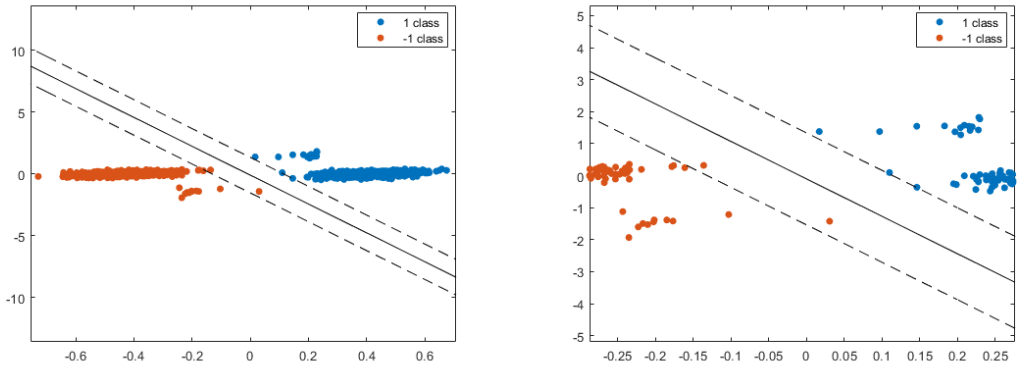


Figure 2.3: The Optimal Hyperplane in SVM

The primal form of SVM is shown as follows

$$\min_{v} \ \frac{1}{2}\|v\|_2^2 + C\sum_{i=1}^{n}\max(0, 1 - y_i v^T x_i), \tag{2.13}$$

where the hinge loss is used to deal with non-separable data where no hyperplane with hard constraints can be found. By introducing slack variables $\{\xi_i \geq 0\}$, we can reformulate the problem (2.13) as constrained optimization problem

$$\begin{aligned} \min_{v,\{\xi_i\}} \ & \frac{1}{2}v^T v + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \ & \xi_i \geq 1 - y_i v^T x_i, \ \xi_i \geq 0, \ \forall i. \end{aligned} \tag{2.14}$$

By using Lagrange multipliers technique, we can further obtain its dual problem according to the duality theorem. First, we construct the Lagrange function with multipliers $\{\alpha_i \geq 0\}_{i=1}^n$ and $\{\tau_i \geq 0\}_{i=1}^n$ as

$$L(v, \{\xi_i\}, \{\tau_i\}, \{\alpha_i\}) = \frac{1}{2}v^T v + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i v^T x_i) - \sum_{i=1}^{n}\tau_i\xi_i. \tag{2.15}$$

Then, the KKT [60] conditions can be obtained by taking differential of L with respect to v, and ξ_i,

$$\frac{\partial L}{\partial v} = 0 \Rightarrow v = \sum_{i=1}^{n}\alpha_i y_i x_i, \tag{2.16}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \tau_i = 0, \tag{2.17}$$

with condition that the primal problem is strongly convex. If v, α_i, and τ_i satisfy the KKT condition, then they are also the solutions to the primal and dual problems [60, 88].

Finally, by substituting v back to Lagrange function as

$$L(\{\xi_i\}, \{\tau_i\}, \{\alpha_i\}) = \frac{1}{2}(\sum_{j=1}^{n} \alpha_j y_j x_j)^T (\sum_{i=1}^{n} \alpha_i y_i x_i)$$

$$+C\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i (1 - \xi_i - y_i (\sum_{i=1}^{n} \alpha_i y_i x_i)^T x_i) - \sum_{i=1}^{n} \tau_i \xi_i, \tag{2.18}$$

$$\Rightarrow L(\{\xi_i\}, \{\tau_i\}, \{\alpha_i\}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + C\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \tau_i \xi_i, \tag{2.19}$$

$$\Rightarrow L(\{\xi_i\}, \{\tau_i\}, \{\alpha_i\}) = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \xi_i (C - \alpha_i - \tau_i). \tag{2.20}$$

Since we have $C - \alpha_i - \tau_i = 0$ from equation (2.17), the dual problem of SVM can be obtained

$$\begin{aligned} \max_{\{\alpha_i\}} \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \alpha_i \\ \text{s.t.} \quad & 0 \le \alpha_i \le C \quad \forall i. \end{aligned} \tag{2.21}$$

Now, we denote $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^T \in \mathbb{R}^n$, and $\boldsymbol{y} = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$. The dual problem of SVM becomes

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})^T X^T X (\boldsymbol{\alpha} \odot \boldsymbol{y}) + \mathbf{1}_n^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n, \end{aligned} \tag{2.22}$$

where $\odot$ is the element-wise product [60].

Let $X(\boldsymbol{\alpha}\odot\boldsymbol{y}) \in \mathbb{R}^d$ be a column vector, then $(\boldsymbol{\alpha}\odot\boldsymbol{y})^T X^T X(\boldsymbol{\alpha}\odot\boldsymbol{y}) = \mathrm{tr}(X(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T X^T)$. By following the trace property, the dual problem of SVM (2.22) is equivalent to

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\mathrm{tr}(X(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T X^T) + \mathbf{1}_n^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n, \end{aligned} \tag{2.23}$$

where $\odot$ is the element-wise product.

CHAPTER 3

A Novel Regularized OPLS Model for Binary Classification

3.1 R-OPLS Binary Classification Model

Based on the review of the previous sections, we can conclude that both least squares and OPLS aim to solve classification problems by minimizing square losses [2, 19]. However, it is important to note that square losses are originally designed for regression problems, not for classification. Therefore, using square losses for classification may not always be the optimal choice, especially when dealing with imbalanced datasets or when the goal is to optimize classification accuracy rather than regression metric, like mean square error.

In order to enhance the accuracy of classification and the generalization of OPLS, we propose a method to jointly optimize OPLS and support vector machines (SVM) by sharing the same projected space in $\mathbb{R}^k$. This approach combines the benefits of both OPLS and SVM to obtain a better model for classification. OPLS can identify the predictive and orthogonal variations in the data, while SVM can provide a robust decision boundary that maximizes the margin between different classes [10, 88]. By sharing the same projected space, we can integrate the strengths of both methods and overcome their respective weaknesses.

The joint optimization of OPLS and SVM can be achieved by minimizing a combination of the squared loss and hinge loss. The squared loss is used to train the OPLS model, while the hinge loss is used to train the SVM model. The weights of the two losses can be adjusted to balance the trade-off between accuracy and robustness.

In this way, the joint optimization can effectively reduce overfitting and improve the generalization of the model.

Moreover, this approach can also handle the situation where the number of samples is smaller than the number of features, which is a common challenge in classification tasks. By projecting the data into a lower-dimensional space, we can reduce the dimensionality of the problem and improve the efficiency of the algorithm.

To achieve this goal, we propose

- SVM trained on the projected space;
- Simultaneous training of OPLS and SVM in the projected space;
- A new optimization approach.

Let's introduce our novel proposed regularized OPLS (R-OPLS). As we mentioned before, let $\{(x_i,\ y_i)\}_{i=1}^n$ be a set of n training samples with c class labels, where $x_i \in \mathbb{R}^d$, and the label $y_i \in \{1, \ldots, c\}$. Denote the input data matrix $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$, and the indicator matrix $Y \in \{0,\ 1\}^{c\times n}$ by using one-hot representation of class labels. Let $\hat{X} = XH \in \mathbb{R}^{d\times n}$, $\hat{Y} = YH \in \mathbb{R}^{c\times n}$ be the centered matrix of X and Y, where $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T \in \mathbb{R}^{n\times n}$ is the centering matrix, I_n is the identity matrix of size n, and $\mathbf{1}_n$ is the n-dimensional column vector of all ones.

The R-OPLS model by trading off square losses and hinge losses is shown as follows

$$\begin{aligned} \min_{P,W}\ & \|\hat{Y} - W^T P^T \hat{X}\|_F^2 - \frac{1}{2}\mathrm{tr}(\Omega^{-1} P^T \hat{X}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T \hat{X}^T P) + \mathbf{1}_n^T\boldsymbol{\alpha} \qquad (3.1)\\ \text{s.t.}\ & P^T P = I_k,\\ & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n, \end{aligned}$$

where $\Omega \in \mathbb{R}^{k\times k}$ is symmetric positive definite, and the coefficient matrix $W \in \mathbb{R}^{k\times c}$.

By following the trace property, the R-OPLS model in problem (3.1) can be further simplified as

$$\max_{\boldsymbol{\alpha}} \min_{P} \; -\mathrm{tr}((P^T(\hat{X}\hat{X}^T+\epsilon I_d)P)^{-1}P^T\hat{X}\hat{Y}^T\hat{Y}\hat{X}^TP+\frac{\lambda}{2}\Omega^{-1}P^T\hat{X}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\hat{X}^TP)+\lambda\mathbf{1}_n^T\boldsymbol{\alpha}.$$

$$\text{s.t.} \;\; P^TP = I_k, \tag{3.2}$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n.$$

3.2 Algorithm

3.2.1 Initialization

To tackle the optimization problem (3.1), we break it down into two sub-problems. The first sub-problem was concerned with finding the unknown variable P, while the second sub-problem focused on determining the unknown parameter $\boldsymbol{\alpha}$. To solve these sub-problems, we leveraged the alternating iterations method, which involves iteratively solving for one variable while holding the other constant, and then switching roles.

To kick off the alternating iterations, we needed to establish an initial value for $\boldsymbol{\alpha}$. To achieve this, we opted to solve the dual form of SVM in the input space. This approach involved formulating and solving the following optimization problem:

$$\boldsymbol{\alpha_0} = \arg\min_{\boldsymbol{\alpha}} \;\; \frac{1}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\hat{X}^T\hat{X}(\boldsymbol{\alpha}\odot\boldsymbol{y}) - \mathbf{1}_n^T\boldsymbol{\alpha} : s.t. \;\; \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n. \tag{3.3}$$

The optimization problem (3.3) is a quadratic programming problem that is subject to a box constraint. Such problems can be efficiently solved using off-the-shelf solvers such as the solver provided by MATLAB [1].

3.2.2 Solving P

3.2.2.1 Generalized Eigenvalue Problem Solver

To streamline the optimization process, we employ the same methodologies elucidated in Subsection 2.2. Additionally, specific choices of Ω can lead to a simplified formulation of problem (3.3) for ease of its numerical treatment. For example, for $\Omega = P^T(\hat{X}\hat{X}^T + \epsilon I_d)P$, the R-OPLS model of problem (3.1) is equivalent to

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \min_{P} \quad & -\mathrm{tr}(P^T\hat{X}[\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T]\hat{X}^T P) + \lambda \mathbf{1}_n^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & P^T(\hat{X}\hat{X}^T + \epsilon I_d)P = I_k, \\ & \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n. \end{aligned} \tag{3.4}$$

Because we changed the SVM primal form to the dual form, the constraint of α_i has been added upper bound of C. Without slack variables $\xi_i > 0$, α_i tends to infinity when the constraints are violated. The upper bound of C limits the α_i and prevents mis-classification.

With initial value of $\boldsymbol{\alpha}$, the problem (3.4) was achieved by solving a particular optimization problem that we denote as problem (3.5).

$$\max_{P} \; tr(P^T Q P) : \text{s.t. } P^T(\hat{X}\hat{X}^T + \epsilon I_d)P = I_k, \tag{3.5}$$

where $Q = \hat{X}(\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T)\hat{X}^T \in \mathbb{R}^{d \times d}$.

Problem (3.5) is a generalized eigenvalue problem [26] on matrix Q with $(\hat{X}\hat{X}^T + \epsilon I_d)$. The optimal solution of P is the eigenvectors of Q with $(\hat{X}\hat{X}^T + \epsilon I_d)$ corresponding to top k eigenvalues [26]. And the Lagrange multiplier is diagonal in the eigenvalue problem [22].

Exploiting the properties of the trace operation, the objective function can take any of the following forms: $\text{tr}(P^TQP)$, $\text{tr}(PP^TQ)$, or $\text{tr}(QPP^T)$, as demonstrated in [6, 26]. Formulating the Lagrangian [22, 27] for problem (3.5), we arrive at

$$L = \text{tr}(P^TQP) - \text{tr}(\Lambda^T(P^T(\hat{X}\hat{X}^T + \epsilon I_d)P - I_k)), \tag{3.6}$$

where $\Lambda \in \mathbb{R}^{d\times d}$ represents a diagonal matrix with entries as the Lagrange multipliers.

To find the optimal P, we need to take the derivative of the objective function L in (3.6) with respect to P and set it equal to zero. This gives us the following equation

$$\frac{\partial L}{\partial P} = 2QP - 2(\hat{X}\hat{X}^T + \epsilon I_d)P\Lambda = 0, \tag{3.7}$$

$$QP = (\hat{X}\hat{X}^T + \epsilon I_d)P\Lambda. \tag{3.8}$$

Then, we also can solve the problem by using

$$(\hat{X}\hat{X}^T + \epsilon I)^{-1}QP = P\Lambda, \tag{3.9}$$

and the diagonal elements of Λ are the eigenvalues [22, 27] .

It is important to note that providing valid values of Ω is necessary to ensure that R-OPLS model in problem (3.1) can be equivalent to the generalized eigenvalue problem. In such cases, finding a numerical solution for equation (3.1) can be a subject of further investigation.

3.2.2.2 Manopt Solver

In Section 3.2.2.1, we introduced a method for solving the matrix P by solving problem (3.1) as a generalized eigenvalue problem. As previously discussed, the transformation of problem (3.1) into a generalized eigenvalue problem for solving the unknown matrix P is not feasible unless we are provided with an appropriate and

effective Ω. In the absence of Ω, an alternative method known as *manopt* can be used to solve the unknown matrix P.

Manopt is a powerful MATLAB toolbox specifically designed for optimization on manifolds [12]. Unlike traditional optimization problems that involve variables in Euclidean space, *Manopt* provides a framework for solving optimization problems where the decision variables reside on smooth manifolds [12]. It offers a wide range of algorithms tailored for optimization on manifolds, including gradient descent, conjugate gradient, trust-region methods, and more [12]. Additionally, *Manopt* supports various types of manifolds, such as spheres, Stiefel manifolds, Grassmann manifolds, and symmetric positive definite matrices, among others [12]. This flexibility makes *Manopt* particularly useful in fields like computer vision, machine learning, robotics, and physics, where manifold optimization problems naturally arise.

When utilizing the *Manopt* toolkit, the first step involves computing the gradient of the objective function presented in problem (3.1). By accurately calculating the gradient, we gain valuable information about the direction of steepest ascent and descent on the manifold. This gradient computation serves as a crucial component for various optimization algorithms provided by *Manopt* [12]. In the gradient calculation, we define matrices $A = \hat{X}\hat{X}^T + \epsilon I_d$, $M = \hat{X}\hat{Y}^T\hat{Y}\hat{X}^T$, and $\Upsilon = \hat{X}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T\hat{X}^T$.

Therefore, the problem (3.1) can be rewritten as

$$\min_P F(P) = -\mathrm{tr}((P^TAP)^{-1}P^TMP + P^T\Upsilon P), \text{ s.t. } P^TP = I_k. \tag{3.10}$$

In [59],

$$\frac{\partial}{\partial P}\mathrm{tr}(P^T\Upsilon P) = \Upsilon P + \Upsilon^T P = 2\Upsilon P, \tag{3.11}$$

$$\frac{\partial}{\partial P}\mathrm{tr}[(P^TAP)^{-1}(P^TMP)] = -2AP(P^TAP)^{-1}P^TMP(P^TAP)^{-1} + 2MP(P^TAP)^{-1}. \tag{3.12}$$

Therefore, we can have

$$\frac{\partial F(P)}{\partial P} = 2AP(P^TAP)^{-1}P^TMP(P^TAP)^{-1} - 2MP(P^TAP)^{-1} - 2\Upsilon P. \quad (3.13)$$

After computing the gradient, we can employ *Manopt* to perform iterative optimization on the manifold. The optimization process involves updating the matrix P based on the computed gradient and a chosen optimization algorithm. The iterative nature of *Manopt* allows for convergence towards an optimal solution by iteratively refining the decision variables on the manifold [12].

Furthermore, *Manopt* offers additional functionalities that enhance the optimization process. These include customizable stopping criteria, which determine when to terminate the iterative procedure based on user-defined thresholds or convergence conditions [12]. Additionally, *Manopt* provides tools for monitoring and visualizing the optimization progress, enabling users to gain insights into the behavior of the algorithm and make informed decisions [12].

The utilization of *Manopt* as a solver provides an alternative approach for solving the matrix P in problem (3.1). By leveraging optimization on manifolds, *Manopt* offers a versatile toolkit for efficiently solving complex optimization problems in various domains [12]. With its wide range of optimization algorithms, support for different manifold types, and additional functionalities, *Manopt* proves to be a valuable resource for researchers and practitioners seeking to tackle optimization challenges on manifolds [12].

3.2.3 Solving $\boldsymbol{\alpha}$

The second sub-problem involves finding the optimal $\boldsymbol{\alpha}$ given P. Once the optimal $\boldsymbol{\alpha}$ is obtained, we can update its value as the new $\boldsymbol{\alpha}$ for the next iteration. Here, we introduce two methods to solve for $\boldsymbol{\alpha}$. The first method is to use a quadratic

programming solver, such as the one that comes with MATLAB [1]. The second method is projected gradient descent(PGD) [8].

3.2.3.1 Quadratic Programming solver (*quadprog*)

As $\hat{P}$ is known, the optimization problem (3.4) can be rewritten as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\mathrm{tr}(P^T\hat{X}[\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T]\hat{X}^TP) + \lambda\mathbf{1}_n^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{3.14}$$

By dropping constant terms, the problem3.14 is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathrm{tr}(P^T\hat{X}[\frac{\lambda}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T]\hat{X}^TP) - \lambda\mathbf{1}_n^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{3.15}$$

Therefore, the optimization problem can be reformulated using the trace properties as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\hat{X}^TPP^T\hat{X}(\boldsymbol{\alpha}\odot\boldsymbol{y}) - \mathbf{1}_n^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{3.16}$$

Then, denoting that $B = (\hat{X}^TPP^T\hat{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T) \in \mathbb{R}^{n\times n}$ is positive definite, we have the following optimization problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^TB\boldsymbol{\alpha} - \mathbf{1}_n^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{3.17}$$

To address problem (3.17), we can employ the same solver used in the initialization phase with a slight modification. The only alteration is that we first need to project the input data onto the low-dimensional subspace $\mathbb{R}^k$.

As a result, we present Algorithm 3.1, which outlines the R-OPLS iterative algorithm utilizing a generalized eigenvalue problem solver and the *quadprog* method. Additionally, Algorithm 3.2 illustrates the R-OPLS iterative algorithm combining the *manopt* solver and the *quadprog* solver.

Algorithm 3.1 R-OPLS Iteration with GEPS+Quadratic Programming Solver

1: **Initialization:** solving $\boldsymbol{\alpha_0}$ by (3.3)
2: **for** i=1 to 50 **do**
3: Update $P^{(i)}$ based on (3.5)
4: Update $\boldsymbol{\alpha}^{(i)}$ based on (3.17)
5: **end for**
6: **Output**: $\boldsymbol{\alpha}$ and P

Algorithm 3.2 R-OPLS Iteration with Manopt+Quadratic Programming Solver

1: **Initialization:** solving $\boldsymbol{\alpha_0}$ by (3.3)
2: **for** i=1 to 50 **do**
3: Update $P^{(i)}$ based on (3.10)
4: Update $\boldsymbol{\alpha}^{(i)}$ based on (3.17)
5: **end for**
6: **Output**: $\boldsymbol{\alpha}$ and P

These algorithms outline the iterative process for R-OPLS, where in each iteration, the estimated matrix P and the coefficient vector $\boldsymbol{\alpha}$ are updated based on specific equations. In Algorithm 3.1, the generalized eigenvalue problem solver and the quadratic programming solver are utilized. On the other hand, Algorithm 3.2

combines the manopt solver with the quadratic programming solver. Both algorithms iterate 50 times to refine the solutions. The outputs include the final values of $\boldsymbol{\alpha}$ and P obtained from the iterations.

3.2.3.2 Projected gradient descent method (PGD)

In addition to using the methods mentioned earlier, we can also use the projected gradient descent method (PGD) [8] to obtain the optimal solution for $\boldsymbol{\alpha}$. This method involves reformulating the problem as follows:

$$\min_{\boldsymbol{\alpha}\in\mathcal{A}} \; g(\boldsymbol{\alpha}) := \max_{P} \; \mathrm{tr}(P^T\hat{X}(\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T)\hat{X}^TP) - \lambda\mathbf{1}_n^T\boldsymbol{\alpha}, \qquad (3.18)$$

where $\mathcal{A}$ represents the domain of $\boldsymbol{\alpha}$, such as $\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n$.

To better understand the PGD method, it's important to first grasp the concept of the gradient descent method [8, 60]. The gradient descent method is a widely used approach for solving unconstrained optimization problems, which involve finding the minimum value of an objective function [8, 20]. At its core, the gradient descent method involves determining the direction of change of the objective function along the direction of the loss function [20]. This iterative process updates the function value continuously, with each iteration reducing the error loss further and moving the learned function closer to the optimal solution of the objective function [8]. The key mathematical concept behind the gradient descent method is the gradient, which involves calculating the partial derivatives of the parameters at a multivariate function and then representing them as a vector [64]. Geometrically, the gradient indicates the direction in which the function changes most rapidly, and moving in the direction of the gradient vector makes it easier to find the maximum value of the objective function [8, 13, 60, 90]. Conversely, moving in the opposite direction of the gradient vector makes it easier to find the minimum value.

In the context of minimizing the loss function, the gradient descent method enables step-by-step iterative solutions to be obtained that minimize the loss function and model parameter values [20]. However, it's important to note that the gradient descent method may not always be able to find the global optimal solution and may only find a local optimal solution [8, 20]. For convex functions, however, the solution obtained by the gradient descent method is always the global optimal solution [60, 90].

The process of gradient descent method:

1) Compute the gradient of the loss function $g(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$ at the current position [64, 90].

$$\nabla_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha}) = \lambda((\hat{X}^T P P^T \hat{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T))\boldsymbol{\alpha} - \lambda \mathbf{1}_n. \tag{3.19}$$

2) Multiply the gradient of the loss function by the step size to get the distance that the current position falls [64, 90].

3) Determine whether all α_i and gradient descent distances are less than ϵ if it is less than ϵ, the algorithm terminates, and the current vector is the final result. Otherwise go to step 4 [64, 90].

4) Update all $\boldsymbol{\alpha}$ can be achieved by performing gradient descent for each α_i, using the following expression. Once the update is complete, proceed to step 1 [64, 90].

$$\boldsymbol{\alpha}^{(t+\frac{1}{2})} \leftarrow \boldsymbol{\alpha}^{(t)} - \lambda \nabla_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha}^{(t)}), \tag{3.20}$$

where $\lambda = \frac{1}{t}$ is the step size of gradient descent method, and t is the iteration number.

In our model, the perform gradient descent expression is

$$\boldsymbol{\alpha}^{(t+\frac{1}{2})} \leftarrow \boldsymbol{\alpha}^{(t)} - \lambda[((\hat{X}^T P P^T \hat{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T))\boldsymbol{\alpha} - \mathbf{1}_n]. \tag{3.21}$$

According to formula (3.21), we can see that λ is not only a tradeoff parameter in our model but is also used in the gradient descent method to update the values of

$\boldsymbol{\alpha}$. It controls the step size during the optimization process and is also referred to as the learning rate in projected gradient descent methods. If the learning rate is too large, the step size of gradient descent may overshoot the minimum point, leading to a failure to converge or even divergence of the objective function [8, 20, 64, 90]. Therefore, it is crucial to choose an appropriate value for λ to ensure the convergence of the optimization process.

In naive gradient descent, the learning rate remains the same. If it is too small, the convergence is slow and it is easy to fall into a local minimum. If it is too large, it is easy to oscillate and fail to converge[8, 20]. Therefore, how to improve the learning rate of the gradient descent method is also an important part of machine learning [20]. In the current scientific research, there are many mature methods for learning rate optimization of gradient descent, such as Stochastic Gradient Descent (SGD) [11], Momentum Gradient Descent (MGD) [61], AdaGrad gradient descent [90], and Root Mean Square Propagation (RMSProp) [49]. In our experiments, the learning rate is equal to $\frac{\lambda}{t}$, where λ is the scaling factor and t is the number of iterations. We introduce the number of iterations into the learning rate. The computation is performed using a method that combines iteration of gradient descent with iteration of alternating solutions. So we only need to adjust the learning rate of gradient descent by tuning the parameter λ.

Projected gradient descent [20, 64, 90] is a commonly used approach for solving constrained optimization problems and is an extension of gradient descent. The method involves computing the update α using the gradient descent approach, which is temporarily denoted as $\alpha^{(t+\frac{1}{2})}$. This update is then project onto the feasible region of α to obtain the next step solution, denoted as α^{t+1}. If the updated α is within the feasible region, then the new $\boldsymbol{\alpha}$ is simply equal to the updated value obtained in the current iteration [60, 64]. However, if the updated α falls outside the bounds

of the feasible region, then the value of α is constrained to lie on the bounds of the feasible region [60, 64]. In our numerical experiments, we set the bounds of α to be between 0 and C, and the projection rule is given by equation (3.22).

- Projection

$$\alpha^{(t+1)} = \begin{cases} \alpha^{(t+\frac{1}{2})}, & \alpha^{(t+\frac{1}{2})} \in [0,\ C]; \\ 0, & \alpha^{(t+\frac{1}{2})} < 0; \\ C, & \alpha^{(t+\frac{1}{2})} > C. \end{cases} \tag{3.22}$$

Based on the projected gradient descent (PGD) approach, we propose two additional algorithms: Algorithm 3.3 and Algorithm 3.4. Algorithm 3.3 outlines the R-OPLS iterative algorithm that combines generalized eigenvalue problem solvers with the PGD method. Similarly, Algorithm 3.4 illustrates the R-OPLS iterative algorithm that combines the manopt solver with the PGD solver.

Algorithm 3.3 R-OPLS Iteration with GEPS+Projected Gradient Descent

Initialization: Solve (3.3) for $\boldsymbol{\alpha_0}$
for i=1 to 50 **do**
    Update $P^{(i)}$ based on (3.5)
    Update $\alpha^{(i+\frac{1}{2})} \leftarrow \alpha^{(i)} - \lambda \nabla_\alpha g(\alpha^{(t)})$
    if $\alpha^{(i+\frac{1}{2})} \in [0,\ C]$ **then**
        $\alpha^{(i+1)} = \alpha^{(i+\frac{1}{2})}$
    else
        if $\alpha^{(i+\frac{1}{2})} < 0$ **then**
            $\alpha^{(i+1)} = 0$
        else
            $\alpha^{(i+1)} = C$
        end if
    end if
end for
Output: $\boldsymbol{\alpha}$ and P

To summarize, the R-OPLS model involves two unknowns: P and $\boldsymbol{\alpha}$. In the solution process for P, we employ two approaches: approximating the model as a generalized eigenvalue problem and directly solving it using the *manopt* toolkit. For the solution process of $\boldsymbol{\alpha}$, we utilize two methods: the *quadprog* solver and the projected gradient descent method. By combining these different approaches for solving P and $\boldsymbol{\alpha}$, we obtain four fundamental algorithms for R-OPLS.

Algorithm 3.4 R-OPLS Iteration with Manopt+Projected Gradient Descent

Initialization: Solve (3.3) for $\boldsymbol{\alpha_0}$
for i=1 to 50 **do**
    Update $P^{(i)}$ based on (3.10)
    Update $\alpha^{(i+\frac{1}{2})} \leftarrow \alpha^{(i)} - \lambda \nabla_\alpha g(\alpha^{(t)})$
    if $\alpha^{(i+\frac{1}{2})} \in [0,\ C]$ **then**
        $\alpha^{(i+1)} = \alpha^{(i+\frac{1}{2})}$
    else
        if $\alpha^{(i+\frac{1}{2})} < 0$ **then**
            $\alpha^{(i+1)} = 0$
        else
            $\alpha^{(i+1)} = C$
        end if
    end if
end for
Output: $\boldsymbol{\alpha}$ and P

By iteratively solving these two sub-problems while updating the values of $\boldsymbol{\alpha}$ and $\hat{P}$ at each step, we were able to converge to a solution for the original problem (3.4). It should be noted that while off-the-shelf solvers can be used to solve quadratic programming problems, the alternating iterations method is often employed when dealing with more complex optimization problems that do not have a closed-form solution. This approach allows for efficient convergence to a solution even when the problem is high-dimensional or the objective function is non-convex.

3.3 Numerical Experiments

3.3.1 Data Information

This section presents experimental results based on three simulation data sets obtained from LIBSVM [16], namely **a2a** [16, 21], **heart** [16, 21], and **w2a** [16, 21]. Table 3.1 provides a summary of relevant data information and parameter settings, including the dimensions of the data sets, the number of training and testing sets. The experiments were conducted on a desktop computer equipped with an Intel(R) Core(TM) i7-4790 CPU and 32-GB RAM, using MATLAB 2021b.

The **a2a** data set is sourced from the UCI Machine Learning Repository Adult Data Set [21], which is based on census data to predict whether an adult earns more than $\$50000/year$ [21]. The original Adult data set comprises 14 features, of which 6 are continuous and 8 are categorical. For the a2a data set, 32561 instances were selected for the experiments. 6 continuous features were discretized into quantiles represented by binary features. and the categorical features were also converted to binary features [16]. As a result, the **a2a** data set has a total of 123 binary features, a total of 2265 instances in the training set, and 30296 instances in the testing set. This information is summarized in Table 3.1.

The **heart** data set, sourced from the UCI Machine Learning Repository [21], provides data on patients with heart disease. It comprises 13 features, such as age, gender, chest pain type (cp), trestbps, serum cholesterol in mg/dl (chol), fasting blood sugar (fbs), resting electrocardiographic results (restecg), maximum heart rate achieved (thalach), exercise-induced angina (exang), ST depression induced by exercise relative to rest (oldpeak), the slope of the peak exercise ST segment (slope), number of major vessels (ca), and thal [21]. The data set comprises 270 instances. To prepare for the experiment, we randomly split the data set into training and test

sets, with 80% of the instances designated for training and the remaining 20% for testing.

The **w2a** data set is derived from the same source as the a2a data set, which is the UCI Machine Learning Repository Adult Data Set [21]. Unlike **a2a**, the **w2a** data set does not discretize continuous features. In the **w2a** data set [16], each data point $x_i \in \mathbb{R}^d$ has 300 features. The total number of training set data instances is 4912, and the total number of testing set data instances is 46279.

Data	**Features**	**Training**	**Testing**	C	λ
a2a	123	2265	30296	0.01, 0.1, 1, 10	0.1, 0.3, 0.5, 0.7, 0.9
heart	13	216	54	0.01, 0.1, 1, 10	0.1, 0.3, 0.5, 0.7, 0.9
w2a	300	4912	46279	0.01, 0.1, 1, 10	0.1, 0.3, 0.5, 0.7, 0.9

Table 3.1: Data Information and Tuning Parameters

3.3.2 Iteration of Objective Function

In the previous sections, we derived that in order to obtain the optimal solution for the unknown variables P and $\boldsymbol{\alpha}$, we need to solve the objective function of the optimization problem to obtain the classifier. To accomplish this, we split the objective function into two sub-problems, one for solving $\boldsymbol{\alpha}$ and another for solving P. The final optimal solution of $\boldsymbol{\alpha}$ and P is obtained through an alternate iteration method. Alternating method needs initialization of variables to start with.

As previously mentioned in the mathematical derivation section, when solving for the unknown variable P using the generalized eigenvalue solver, we need to introduce regularization to avoid the problem of $\hat{X}\hat{X}^T$ being a singular matrix. This can be achieved by adding a small positive number ϵ to $(\hat{X}\hat{X}^T + \epsilon I_d)$, where I_d is the identity matrix of size d. For our experiments, we set ϵ to be 10^{-8} for all data sets to ensure that $\hat{X}\hat{X}^T + \epsilon I_d$ is of full rank.

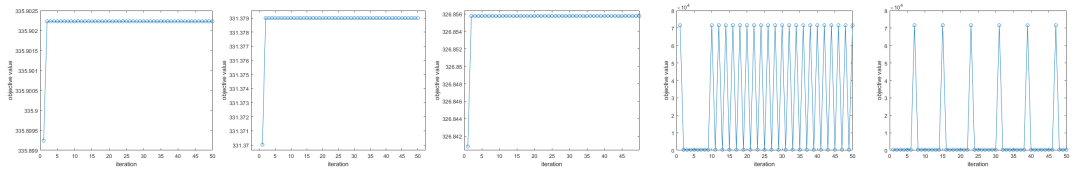
There are two methods we use to solve for the unknown α. The first method is using a built-in MATLAB function called *quadprog* [1], which is a popular solver for quadratic problems. *quadprog* also has parameters such as *Algorithm* [1], *MaxIterations* [1], and *OptimalityTolerance* [1]. The default *Algorithm* used by MATLAB is *interior-point-convex* [1], the default *MaxIterations* is 200, and the default *OptimalityTolerance* is 10^{-8} [1]. The second method we use is the projected gradient descent method mentioned earlier. Through iterative plots of the objective function value, we found that the convergence of the gradient descent method is better than that of *quadprog* in our experiments. We will provide more specific details in the following section.

The R-OPLS model is governed by three parameters: C, λ, and k. Since the optimal problem is min-max problem, the objective function value may oscillate slightly within a small range once it approaches convergence. Using the parameters C, k, and λ, we plot the objective function value curve after projecting the data onto a k-dimensional space.

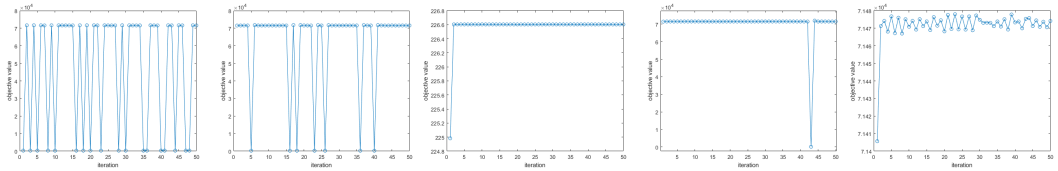
During the numerical experiments, we conducted 20 repetitions for all cases and tested different values of the parameters C, k, and λ. Specifically, we explored the range of values $C \in \{10^{-2}, 10^{-1}, 1, 10\}$, $k \in \{3, 10, 20\}$, and $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

As a result, the oscillation amplitudes of the iterative calculation results of the objective function can vary significantly under different penalty parameters. In particular, when we keep the value of k constant and change the values of C and λ, the convergence of the objective function value changes significantly. Figure 3.1 depicts the change in the function value of the objective function under different values of C and λ in iterative experiments. Each row represents the function value change under different C values, while each column represents the function value change under different λ values. For instance, the first row represents the function

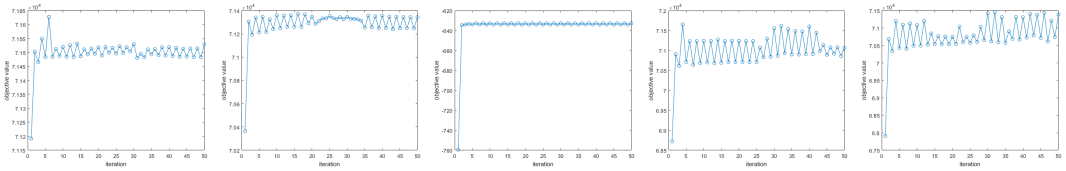
value change when $C = 0.01$, the second row when $C = 0.1$, the third row when $C = 1$, and the fourth row when $C = 10$. Similarly, the first column represents the function value change image when $\lambda = 0.1$, the second column when $\lambda = 0.3$, the third column when $\lambda = 0.5$, the fourth column when $\lambda = 0.7$, and the fifth column when $\lambda = 0.9$.



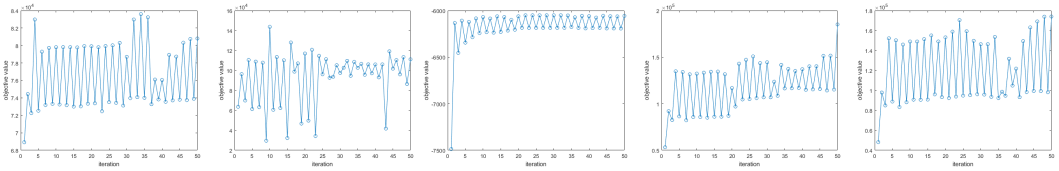
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 3.1: The **a2a** Data Set with *quadprog* Solver

It is apparent that there exists a noteworthy correlation between the value of C and the amplitude of the oscillations of the function's plots. As C increases, the oscillations of the function image become more pronounced. Specifically, when C is

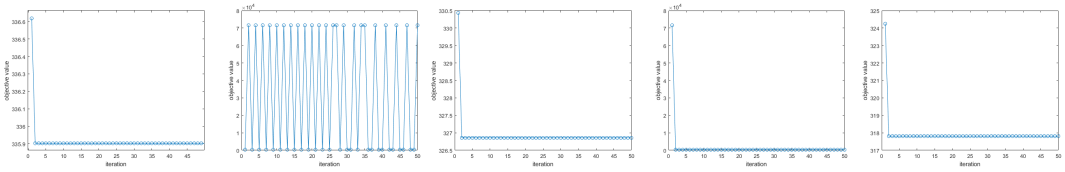
set to 10, the amplitude increases significantly. This trend implies that large values of C may result in more significant fluctuations in the convergence of the objective function. Therefore, care must be taken when selecting a value for C to ensure that the objective function converges appropriately. We can observe that there is a considerable variation in the convergence of the objective function for different values of λ when C is large. This observation suggests that the selection of an appropriate value for λ is particularly crucial when working with large C values. By observing Figure 3.1, it is only when λ is set to 0.5 that the function image maintains adequate convergence for various C values.

As previously mentioned, the projected gradient descent method also can be employed to solve $\boldsymbol{\alpha}$ in the objective function. Examining the formula $\nabla_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha}) = \lambda((\hat{X}^T P P^T \hat{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T))\boldsymbol{\alpha} - \lambda \mathbf{1} n$, we can observe that the penalty parameter C and scale factor λ directly affect the gradient descent rate $\nabla \boldsymbol{\alpha} g(\boldsymbol{\alpha})$.

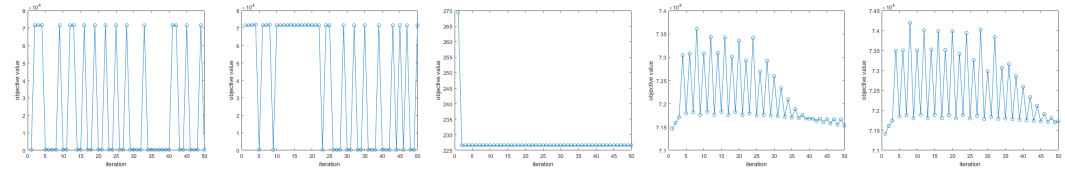
It is crucial to select appropriate values for the penalty parameter C and scale factor λ when implementing the gradient descent method. Increasing the penalty parameter C will result in an increase in the value range of $\boldsymbol{\alpha}$. The penalty parameter C plays a crucial role in controlling the gradient's magnitude and direction, as illustrated in the mathematical derivation provided earlier. On the other hand, the parameter λ is responsible for controlling the step size in the PGD process.

It is essential to select an appropriate value for λ since if it is too large, the step size of PGD may exceed the minimum point. Consequently, the objective function may fail to converge or, worse still, diverge. Thus, choosing the right value for the learning rate λ is crucial for ensuring the successful convergence of the objective function during PGD.

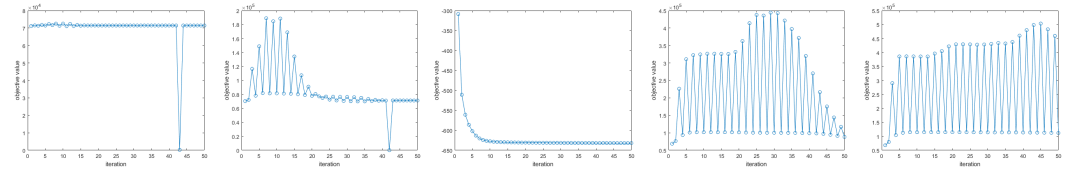
Figure 3.2 provides insights into the impact of varying C on the gradient descent curves' convergence. When C is set to 0.01 or 0.1, the descending convergence



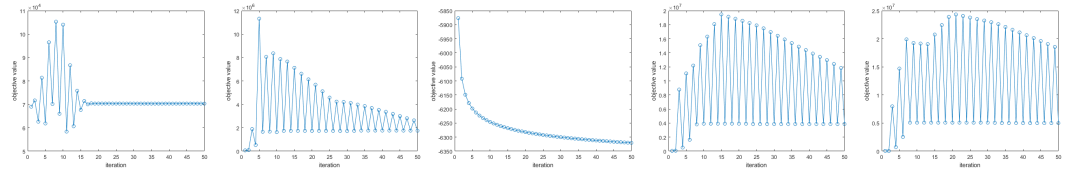
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 3.2: The **a2a** Data Set with PGD Method

curves in Figure 3.2 (a)-(j) display varying degrees of stagnation. On the other hand, Figure 3.2(m) and Figure 3.2(r) illustrate that when $\lambda = 0.5$ and C is set to 1 or 10, the gradient descent curves exhibit smooth convergence. Selecting the appropriate values for the penalty parameter C and scale factor λ is crucial for achieving desirable results with the projected gradient descent method. The selection process must balance the gradient descent rate's speed and the potential for stagnation in the objective function.

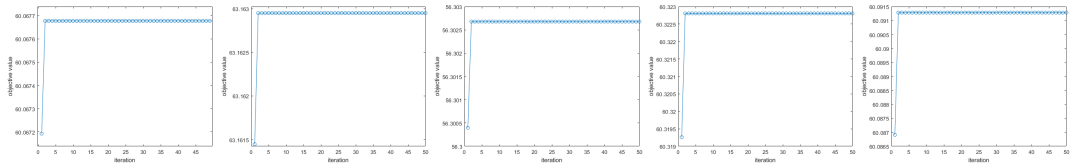
The **heart** data set contains 13 features. To reduce the dimensionality of the heart data, we set the subspace parameter k to 6. In the next step of the experiment,

we studied the behavior of the objective function's iterative curve using the quadprog solver. The results are presented in Figure 3.3.

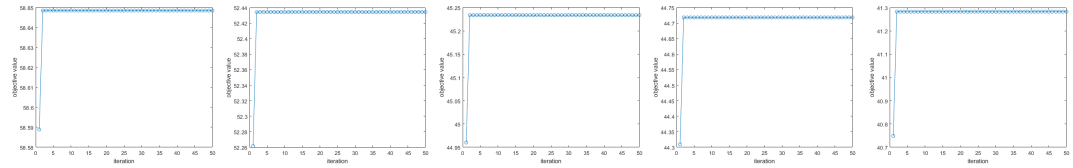
The Figure 3.3 shows that when the penalty parameter C is too small, such as $C = 0.01$ and $C = 0.1$, the change of the scaling factor λ has little effect on the convergence of the objective function. This is because the value range of $\boldsymbol{\alpha}$ is too small to influence the iteration result significantly. However, for larger penalty parameters, such as $C = 1$ and $C = 10$, the choice of scaling factor λ becomes crucial. As we can see from the plots (k)-(t) in Figure 3.3, only when $\lambda = 0.5$ and $C = 1$, the objective function's iterative curve maintains smooth convergence with a slight oscillation, and the oscillation amplitude is within ± 0.5. It is important to note that this conclusion is drawn based on the use of the quadprog solver and may differ for other solvers or optimization algorithms.

Our experiment further suggests that the choice of penalty parameter C and scaling factor λ is critical in achieving the optimal convergence rate of the objective function. Further experiments may be necessary to verify the robustness of these conclusions under different solvers and optimization algorithms.

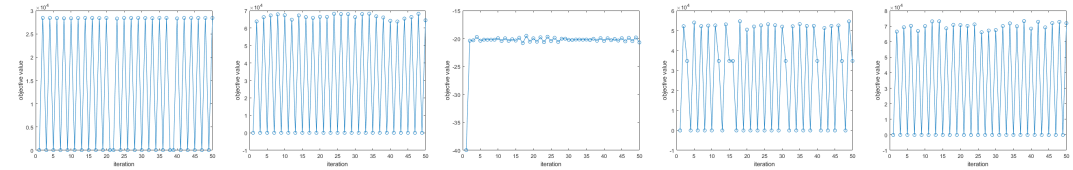
In line with the previous experimental results on the heart data set using the *quadprog* solver, we conducted further experiments using the gradient descent method. We found that when the value of C is too small, the function value is not affected by any changes in the parameter λ. This observation is shown in Figure 3.4 (a)-(j). Furthermore, by comparing the iterative graphs in Figure 3.4 (k)-(t), we discovered that when $\lambda = 0.5$ with $C = 1$ or $C = 10$, the gradient descent method achieves smooth convergence. Therefore, we can conclude that the optimal λ for the R-OPLS model in the heart data experiment is $\lambda = 0.5$, as confirmed by the iterative graph.



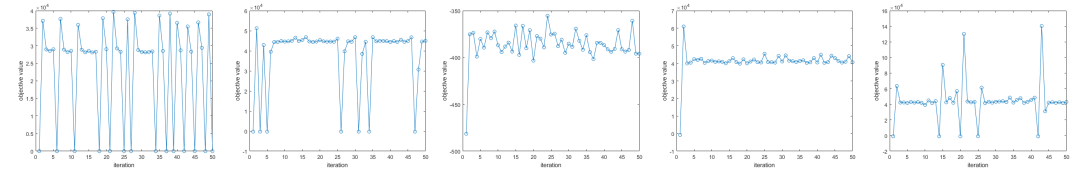
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$

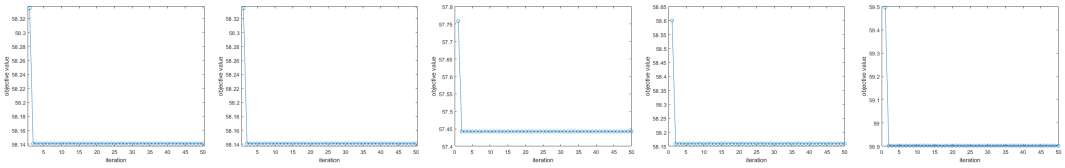


(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

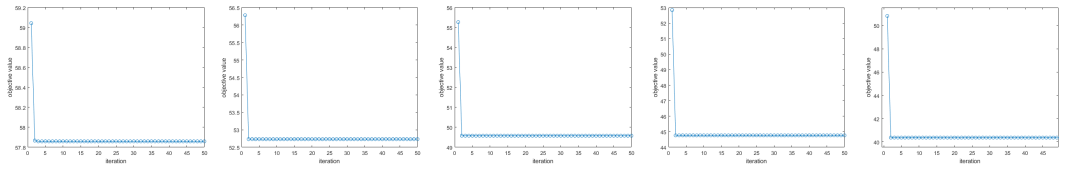
Figure 3.3: The **heart** Data Set with *quadprog* Solver

The **w2a** data set is the largest among our three binary classification experiment data sets, consisting of 4912 training instances and 46279 testing instances. To solve the objective function, we first set the projection space to 10 and adjusted two other parameters, namely C and λ. We explored a range of values for λ, namely 0.1, 0.3, 0.5, 0.7, and 0.9, and set C to a series of 0.01, 0.1, 1, and 10. Similar to the previous experiments, we observed significant influence of parameter changes on the function value in alternate iterations, as depicted in Figure 3.5.

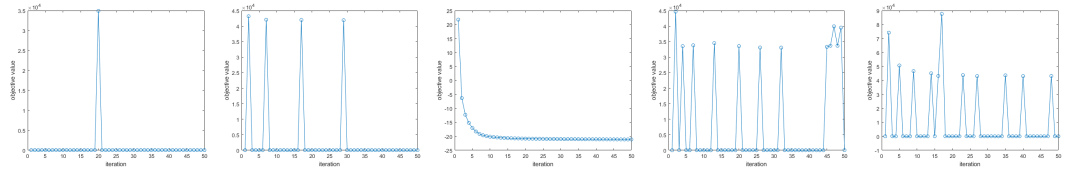
Based on the observations made from Figure 3.5, we can conclude that when the value of the regularization parameter C is set to 0.01, the first four groups of



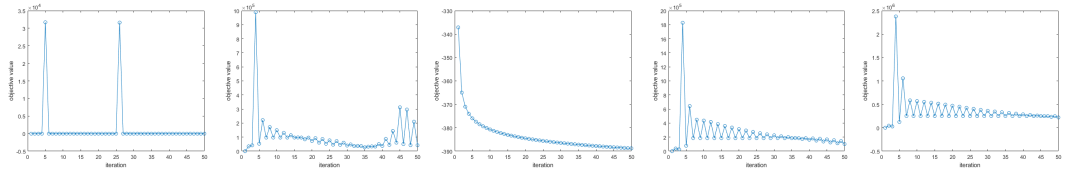
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



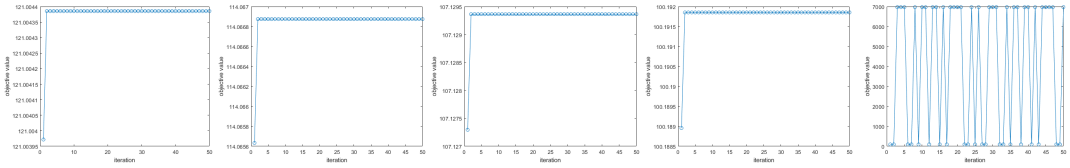
(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



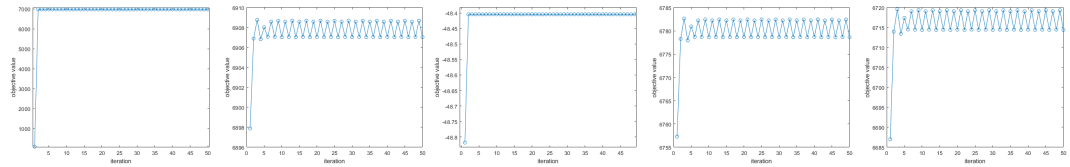
(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 3.4: The **heart** Data Set with PGD Method

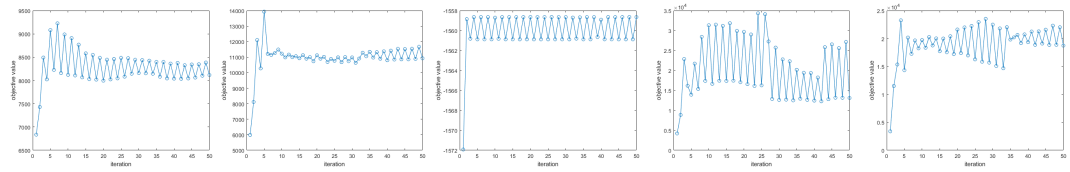
curve images do not show significant changes. However, in the last group where $C = 0.01$ and $\lambda = 0.9$, the function value exhibits significant oscillations in the iterative experiment, with an oscillation range of $0 - 7000$. This suggests that, regardless of how λ changes, a C value of 0.01 is not a reasonable parameter setting for the R-OPLS model based on w2a data. On the other hand, when the value of C is increased to 0.1, we observe that the plots of (g), (i), and (j) exhibit small oscillations with an oscillation range of 69083, 67803, and 67173 respectively. As the value of C further increases to 1 and 10, we notice a significant increase in the



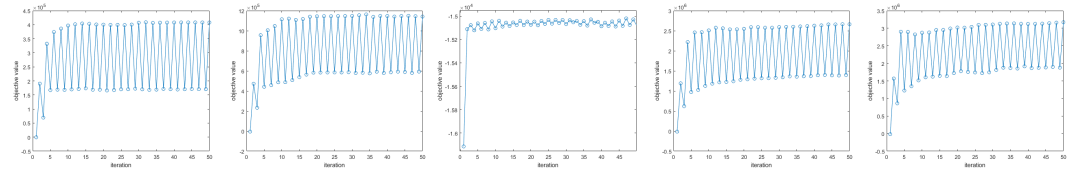
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 3.5: The **w2a** Data Set with *quadprog* Solver

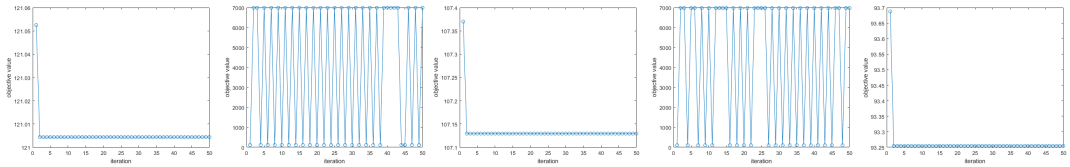
oscillation amplitude of the function value in the iterative experiment, as shown in plots (k)-(t).

To determine the optimal parameters for the w2a experiments, we adopted an iterative approach using projected gradient descent to solve $\boldsymbol{\alpha}$. This method allowed us to experiment with various parameter settings and select the best ones based on the model's performance.

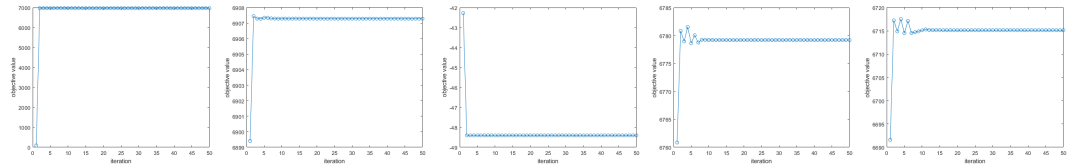
To replicate the experiment, we began by initializing the projection space to 10 and tuning the other two parameters, C and λ. The range of values for λ and C were identical to the previous experiment. Specifically, we set λ to vary from 0.1 to

0.9 at intervals of 0.2 and C to be one of four values: 0.01, 0.1, 1, and 10. To observe the impact of changing parameters on the objective function value during alternate iterations, we plotted the results in Figure 3.6.

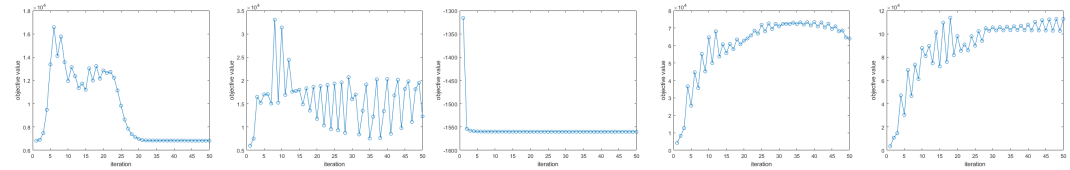
We can see that the first row of plots when $C = 0.01$, the second row of plots when C=0.1, and the third row of plots when C=1 did not show good convergence. This confirms our previous conclusions that when the parameters of the projected gradient descent method are not set properly, it may lead to a situation of stuttering or crossing the minimum value, resulting in shocks.



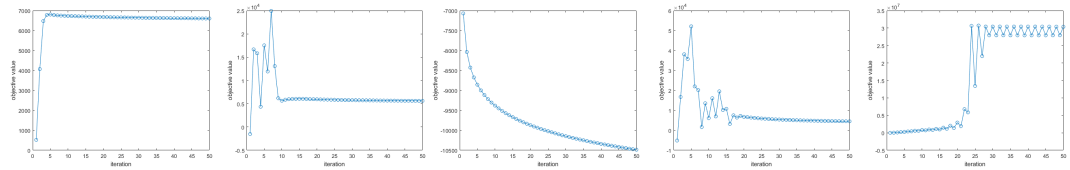
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 3.6: The **w2a** Data Set with PGD Method

However, when $C = 10$ and $\lambda = 0.5$, the function value graph reaches a smooth descent. It is worth noting that when the number of iterations is set to 50, the function value continues to decline and has not reached the minimum convergence value. To address this issue, we increased the number of iterations to 100, 200, 500, and 1000, respectively. However, the function value still kept decreasing. This indicates that the step size of each descent is too small due to the problem of parameter setting, so the projected gradient descent method cannot reach the minimum value required by the experiment in a short time. When $\lambda = 0.7$, although the function value fluctuated in the first 16th iterative experiments, it maintained a downward trend of shock. From the 17th iterative calculation, the function graph began to converge. Based on the above analysis, we can conclude that the projected gradient method is suitable for the w2a data classification experiment, and the optimal parameter selection is $C = 10$ and $\lambda = 0.7$.

It is worth noting that the choice of parameters greatly affects the convergence and accuracy of the algorithm. Therefore, careful experimentation with different parameter settings is necessary to obtain the optimal parameters for each data set.

3.3.3 Projection Classification and Accuracy

The R-OPLS model is a powerful machine learning algorithm that aims to learn a projection matrix $P \in \mathbb{R}^{d \times k}$ to transform input data from a high-dimensional space (R^d) to a low-dimensional space (R^k), where classification is conducted. However, determining the optimal value of k for each data set is still a critical challenge that must be addressed.

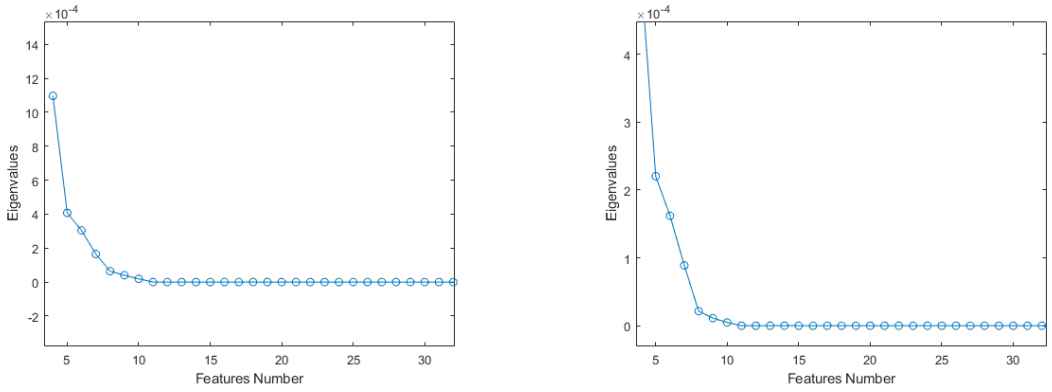
As discussed in earlier sections, the optimal projection matrix P can be computed by identifying the eigenvectors of the matrix Q, where $Q = \hat{X}[\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T]\hat{X}^T \in \mathbb{R}^{d \times d}$. Here, $\hat{X}$ and $\hat{Y}$ are the mean-centered input and output

data matrices, $\boldsymbol{\alpha}$ is a vector of regression coefficients, $\boldsymbol{y}$ is the output variable, and λ is a regularization parameter. After obtaining the optimal solutions for P and $\boldsymbol{\alpha}$, we can construct two classifiers, W and v. The classifier W is the OPLS classifier and can be computed as $W = (P^T\hat{X}\hat{X}^TP)^{-1}P^T\hat{X}\hat{Y}^T$. The classifier v is the support vector machine (SVM) classifier and can be computed as $v = \sum_{i=1}^{n} \alpha_i y_i P^T\hat{x}_i$. Using these classifiers, we can make predictions in the common subspace and evaluate the classification accuracy. This accuracy serves as a metric to assess the performance of the R-OPLS model and its ability to learn a suitable subspace for classification tasks.

$$\text{Classification Accuracy} = \frac{\text{Total Number of Correct Prediction}}{\text{Total Number of Testing Data Points}} \times 100\%$$

To identify the optimal projected subspace dimension k for a given dataset, we first analyze the eigenvalue images of the objective function generated during alternating iterative experiments. This analysis helps us determine the potential range of values for k. The process of selecting an appropriate value for k is crucial for achieving optimal classification accuracy. If k is too small, important information in the data may be lost during projection. Conversely, if k is too large, the classification task may become computationally expensive [60].

Once we have identified the range of potential values, we perform classification experiments for each value of k within this range and evaluate the resulting classification accuracy. We then select the value of k that produces the highest accuracy as the optimal projected subspace dimension for the given data set. This approach allows us to systematically explore the space of possible subspace dimensions and select the dimension that yields the best classification performance. By reducing the dimensionality of the data while preserving the most relevant information, we can improve the accuracy and efficiency of our classification models.



(a) *quadprog* Solver (b) PGD

Figure 3.7: The Plots of Eigenvalues on **a2a** Data

In the **a2a** data set, the two plots depicted in Figure 3.7, namely plot (a) and plot (b), display the eigenvalues of the objective function obtained through the use of the generalized eigenvalue solver. Plot (a) shows the results obtained by combining the solver with the *quadprog* solver, while plot (b) displays the results obtained through the use of the projected gradient descent method in an alternating iteration process. By analyzing the curves shown in Figure 3.7, it can be observed that the changes in eigenvalues tend to flatten out after $k = 8$ in both plots. Hence, we restrict the domain of k to $7 \leq k \leq 11$. Furthermore, in the previous experiments using the alternating iteration approach with either the quadprog solver or the projected gradient descent method, we observed that the function image converged at C=1 and C=10 when the parameter λ was set to 0.5. To ensure the reliability and validity of the experiments, we performed classification experiments at $C = 1$ and $C = 10$, respectively, and compared their classification accuracy results. The classification accuracy results for $C = 1$ and $C = 10$ are presented in TABLE 3.2 below.

Based on the experimental results, we can conclude that the R-OPLS model based on the a2a data set achieves the highest accuracy of 83.75% when using the

	$C = 1$				$C = 10$			
	quadprog		PGD		*quadprog*		PGD	
k	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**
$k = 7$	83.75%	71.68%	78.64%	71.80%	71.04%	61.86%	75.98%	71.81%
$k = 8$	76.02%	71.68%	80.92%	71.80%	25.44%	61.86%	79.42%	71.81%
$k = 9$	76.02%	71.68%	83.22%	71.80%	24.04%	61.86%	76.63%	71.81%
$k = 10$	76.08%	71.68%	77.21%	71.80%	76.01%	61.42%	83.22%	71.81%
$k = 11$	81.84%	71.79%	37.65%	71.80%	81.58%	60.86%	83.31%	71.81%
$k = 12$	76.70%	71.73%	77.24%	71.80%	24.04%	60.86%	81.63%	71.81%

Table 3.2: Accuracy with Selection of k on a2a Data

quadprog solver and the parameter combination of $C = 1,\ \lambda = 0.5,\ k = 7$. Therefore, the experimental results suggest that the R-OPLS model is suitable for the a2a data set when using the quadprog solver with the aforementioned parameter combination.

In our previous experiments, we utilized the generalized eigenvalue solver in alternate iterations to solve for matrix P and evaluate the classification accuracy of R-OPLS on the **a2a** data set. As described in the algorithm section, an alternative approach is to employ the manopt solver for directly solving matrix P within the alternating iterative algorithm. With this in mind, we conducted additional experiments on the **a2a** data set by following the aforementioned steps and using the optimal solution of P obtained through the *manopt* solver.

For these experiments, we set the projection subspace to $k = 7$. The results of these experiments are presented in Table 3.3, providing an insightful comparison between the performance of the generalized eigenvalue solver and the *manopt* solver in the classification task.

When comparing the results presented in Table 3.2 and Table 3.3, a noticeable trend emerges: the classification accuracy achieved by utilizing the *manopt* solver to solve for matrix P diminishes across various parameter combinations. This observation indicates that, in terms of optimizing the R-OPLS model, the *manopt* solver

		manopt + *quadprog*				*manopt*+PGD			
λ	Classifier	$C = 0.01$	$C = 0.1$	$C = 1$	$C = 10$	$C = 0.01$	$C = 0.1$	$C = 1$	$C = 10$
$\lambda = 0.1$	R-OPLS	70.24%	72.74%	67.31%	64.57%	70.67%	70.96%	69.84%	70.36%
	SVM	70.69%	56.76%	62.93%	62.56%	70.49%	71.76%	29.34%	30.12%
$\lambda = 0.3$	R-OPLS	70.04%	71.59%	66.96%	69.98%	72.45%	71.19%	64.25%	63.15%
	SVM	70.09%	68.47%	50.98%	69.94%	72.24%	72.34%	51.28%	56.52%
$\lambda = 0.5$	R-OPLS	71.02%	71.37%	71.44%	70.22%	71.95%	69.63%	71.50%	66.97%
	SVM	70.56%	70.51%	58.92%	69.96%	72.51%	72.18%	72.00%	44.09%
$\lambda = 0.7$	R-OPLS	71.93%	72.22%	64.33%	67.84%	71.95%	69.54%	69.33%	63.88%
	SVM	70.27%	47.08%	56.46%	67.06%	72.51%	29.36%	29.35%	42.60%
$\lambda = 0.9$	R-OPLS	71.03%	70.88%	67.43%	65.73%	70.75%	71.68%	69.52%	69.74%
	SVM	69.91%	70.70%	58.40%	64.84%	72.29%	72.30%	29.89%	71.17%

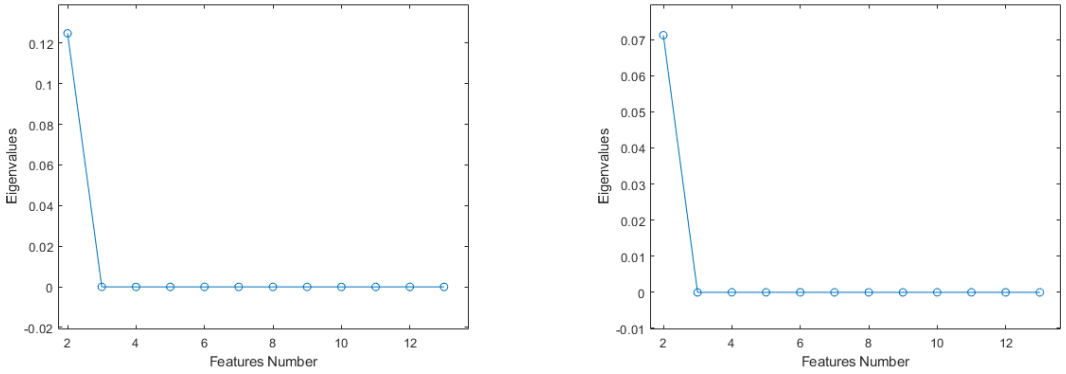
Table 3.3: Classification Accuracy of a2a Data with *manopt* Solver

is not the most favorable choice. Moreover, it is worth highlighting that both the iterative algorithm of *manopt* combined with *quadprog* and the iterative algorithm of *manopt* combined with PGD exhibit longer computational costs compared to the iterative calculations performed using the generalized eigenvalue solver.

Overall, it becomes evident that employing the *manopt* solver in the alternating reception framework necessitates three times the computational cost required when using the generalized eigenvalue solver. Hence, considering both the decrease in classification accuracy and the increased computational burden, the *manopt* solver is not a preferable option for the R-OPLS model. Therefore, for the coming numerical experiments, we will not use *manopt* solver in the alternating iteration experiments.

In the **heart** data set, Figure 3.8 includes two plots, labeled (a) and (b), which depict the eigenvalues of the objective function using different solvers. Specifically, plot (a) shows the eigenvalues when the generalized eigenvalue solver is combined with the *quadprog* solver, while plot (b) displays the eigenvalues when the generalized eigenvalue solver is combined with the projected gradient descent method in an

alternating iteration process. Both figures show that the eigenvalue variations tend to flatten after $k = 3$. Thus, we limit the range of k to $2 \leq k \leq 5$ and compute the classification accuracy for each k. As the heart data set has only 270 cases, we randomly select 80% of the data as the training set and the remaining 20% as the test set for the classification experiment. We perform experiments 15 times with different random data splits to obtain the average classification accuracy. The results of the classification experiments are presented in Table 3.4.



(a) *quadprog* Solver (b) PGD

Figure 3.8: The Plots of Eigenvalues on **heart** Data

From the results presented in Table 3.4, it is evident that the highest average classification accuracy of the R-OPLS model for the heart data set is obtained when using the projected gradient descent method with the parameter combination of $C = 10$, $\lambda = 0.5$, and $k = 3$. Specifically, the average classification accuracy of the R-OPLS model is 85.94%, and the classification result of the SVM classifier in the R-OPLS model reaches a maximum of 86.06%. Therefore, based on these findings, we selected the parameter combination of $C = 10$, $\lambda = 0.5$, and $k = 3$ for the R-OPLS model in the classification experiments on the heart data set.

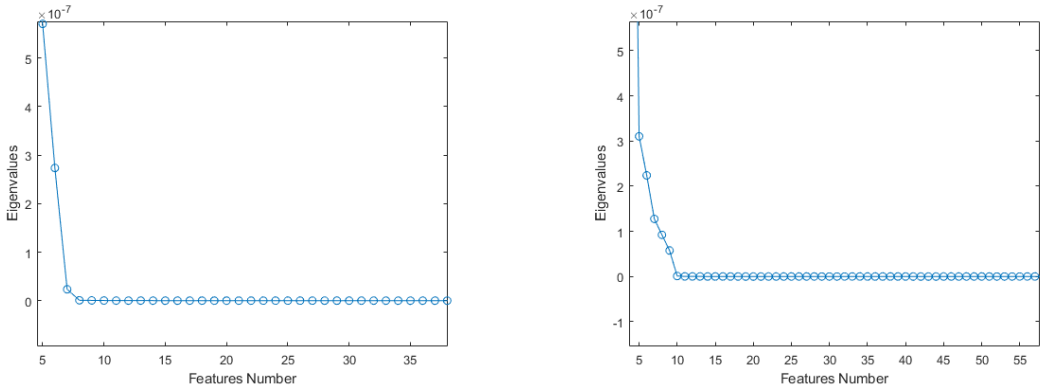
		quadprog		PGD	
	K	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**
$C = 1$	$k = 2$	$83.15 \pm 6.37\%$	$82.42 \pm 6.06\%$	$82.67 \pm 6.31\%$	$83.03 \pm 6.67\%$
	$k = 3$	$82.30 \pm 5.94\%$	$81.57 \pm 5.21\%$	$84.48 \pm 4.61\%$	$84.24 \pm 6.67\%$
	$k = 4$	$83.88 \pm 7.52\%$	$83.15 \pm 8.60\%$	$82.79 \pm 6.43\%$	$83.03 \pm 7.88\%$
	$k = 5$	$82.18 \pm 6.91\%$	$82.18 \pm 5.82\%$	$84.60 \pm 8.24\%$	$85.21 \pm 8.85\%$
$C = 10$	$k = 2$	$83.51 \pm 7.40\%$	$76.97 \pm 4.24\%$	$84.00 \pm 8.73\%$	$83.15 \pm 9.58\%$
	$k = 3$	$84.61 \pm 7.28\%$	$78.30 \pm 7.39\%$	$85.94 \pm 5.94\%$	$86.06 \pm 4.24\%$
	$k = 4$	$83.15 \pm 7.19\%$	$77.82 \pm 6.91\%$	$84.85 \pm 7.88\%$	$84.36 \pm 8.37\%$
	$k = 5$	$82.30 \pm 6.79\%$	$78.79 \pm 8.48\%$	$84.12 \pm 5.94\%$	$84.24 \pm 6.06\%$

Table 3.4: Accuracy with selection of k on **heart** data

The original data sets for w2a and a2a are both from the adult data set, so their effective eigenvalues are similar. In the **w2a** data set, Figure 3.9 presents two plots that display the eigenvalues of the objective function obtained through different methods. In plot (a), we use the generalized eigenvalue solver in combination with the *quadprog* solver, while in plot (b), we use the projected gradient descent method in an alternating iteration process. By observing the curves in FIGURE 3.9, we can see that the eigenvalue changes flatten out after $k = 8$ in both plots. Therefore, we selected the range of k as $7 \leq k \leq 12$ and obtained the classification accuracy for each k, as shown in Table 3.5.

	$C = 1$				$C = 10$			
	quadprog		PGD		*quadprog*		PGD	
k	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**	**R-OPLS**	**SVM**
$k = 7$	86.22%	69.58%	85.80%	70.32%	86.05%	79.72%	86.11%	94.85%
$k = 8$	85.85%	69.69%	86.41%	70.32%	85.81%	79.58%	85.98%	94.85%
$k = 9$	86.07%	69.68%	85.71%	70.32%	85.96%	79.57%	85.74%	94.85%
$k = 10$	85.82%	69.57%	85.74%	70.32%	86.16%	79.06%	85.55%	94.85%
$k = 11$	86.64%	69.18%	85.78%	70.32%	85.99%	79.05%	85.55%	94.85%
$k = 12$	85.83%	69.18%	85.65%	70.32%	85.98%	79.20%	84.33%	94.85%

Table 3.5: Accuracy with Selection of k on w2a Data



(a) "*quadprog*" Solver (b) Projected Gradient Descent

Figure 3.9: The Plots of Eigenvalues on **w2a** Data

From Table 3.5, we can see that the highest average classification accuracy of the R-OPLS model for the w2a data set was achieved with the parameter combination "$C = 1$, $\lambda = 0.5$, and $k = 11$" using the quadprog solver. The classification accuracy reached 86.64%, which indicates that the R-OPLS model is effective for the w2a data set.

3.3.4 Comparison Methods

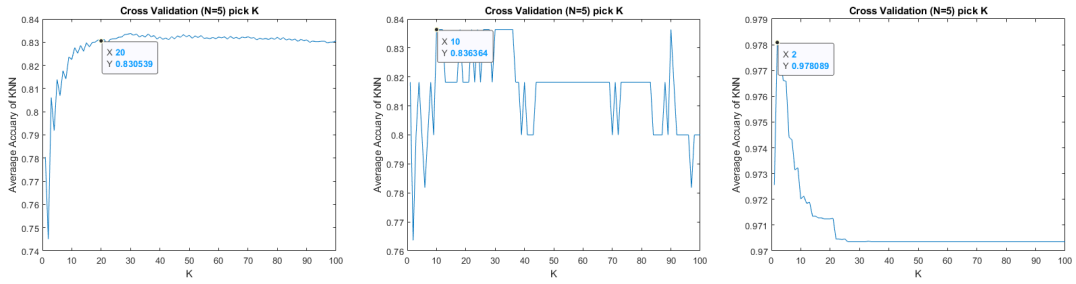
Achieving high classification accuracy is the primary goal of our R-OPLS model. In subsection 3.3.3, we presented the results of the R-OPLS model classification experiments on three datasets. While there are many established models and algorithms available for data classification, we have chosen to use the least squares method, SVM, and K-Nearest Neighbors (KNN) [40] for comparison purposes. In addition to briefly introducing KNN as a classification method, we also explain its working principle. KNN is a simple machine learning algorithm used for supervised classification. When predicting the class of an input data point, KNN determines the class with the highest frequency among the K closest data points, where K is

the number of data points closest to the input point. The majority rule is used to determine the class of the input data point based on the K closest data points.

To use the KNN algorithm for classification prediction, we first use the $fitknn$ [40] function to create a KNN prediction model. This function has several parameters, including $BreakTies$, $BucketSize$, $CategoricalPredictors$, $NumNeighbors$, and $Distance$ [1, 40]. We have modified two of these parameters: $NumNeighbors$ and $Distance$. The $Distance$ parameter has various options, such as *cityblock*, *chebychev*, *correlation*, *cosine*, *Euclidean*, and *mahalanobis* [36]. The specific choice of the $Distance$ parameter depends on the experimental purpose and data type. In our experiments, we set the Distance parameter to Euclidean for all data sets.

The KNN model is highly sensitive to the distribution of data points in space, which can significantly affect the accuracy of its classification results. Selecting the appropriate K value is a crucial step in the KNN algorithm, as it has a direct impact on the classification accuracy. In order to determine the optimal K value for the KNN algorithm on each data set, we employed a 5-folder cross-validation method, selecting K values ranging from 1 to 100. The data was randomly partitioned into 5 parts, with 4 parts as the training set and the remaining part as the test set. The average classification accuracy under each K value was obtained by repeating this process 5 times, and a graph of the classification accuracy rate under each K value was plotted (see FIGURE 3.10)

In the a2a data set, the KNN model showed significant improvement in classification accuracy with increasing K value, reaching about 83% before $K = 20$. After that point, there was little change in the classification accuracy. On the other hand, in the heart dataset, the KNN model achieved a classification accuracy of 83.64% at $K = 10$. In Figure 3.9 (b), the impact of K value on the classification accuracy



(a) a2a Data Set (b) heart Data Set (c) w2a Data Set

Figure 3.10: The Classification Accuracy of KNN

of the KNN model is evident. As K value continues to increase, the classification accuracy of KNN fluctuates up and down. For the w2a dataset, the KNN model achieved its highest classification accuracy of 97.81% at $K = 2$, but the accuracy rapidly declined with increasing K value and finally stabilized at around 97%. These results demonstrate that the KNN model has significant limitations and requires careful data selection and K value selection.

It is worth noting that the KNN model cannot be used to reduce the dimensionality of data, which is one of its major shortcomings. The novel R-OPLS has significant advantages in projection space classification. It is important to mention that the data used in these experiments were all from LIBSVM[16]. Although these data are highly suitable for support vector machine classification, the classification results of support vector machine do not show high accuracy.

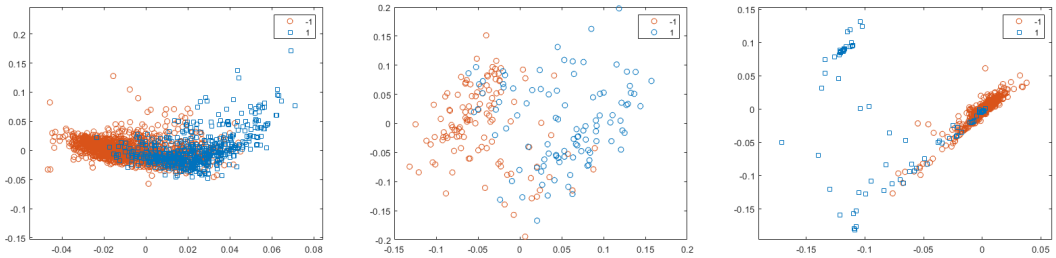
Moving forward, we plan to test the stability and accuracy of our novel R-OPLS model using other dimensionality reduction data, thus demonstrating its advantages in classification on projected spaces. In summary, the KNN model is highly sensitive to data distribution and requires careful parameter selection. The novel R-OPLS model has significant advantages over the KNN model in projection space classification.

Classifiers / Data	Input Space			Projected Space		
	KNN	**SVM**	**Least Squares**	**KNN**	**SVM**	**R-OPLS**
a2a	83.23%	84.32%	72.15%	83.86%	75.96%	83.35%
heart	83.64%	72.73%	80.00%	92.73%	86.06%	85.94%
w2a	97.81%	97.65%	85.83%	98.07%	69.18%	86.64%

Table 3.6: Accuracy Comparison

3.3.5 Visualization

The R-OPLS model serves a crucial function in effectively reducing the dimensionality of data. However, we also need to visually observe the data distribution to improve our model's classification accuracy. Thus, we project each set of data into 2D and 3D spaces, respectively, to intuitively observe their projections in low-dimensional space. Specifically, for each data set, we select $k = 2$ for 2D space and $k = 3$ for 3D space. We combine the optimal values of C and λ obtained from prior experiments and employ the *gscatter* [1] function in MATLAB. FIGURE 3.11 depicts the 2D visualization, with the positive class represented by the blue circle and the negative class represented by the orange circle.



(a) a2a Data Set (b) heart Data Set (c) w2a Data Set

Figure 3.11: The Visualization on 2-D Space

Figure 3.11(a) illustrates that the **a2a** data set is not linearly separable. As shown in the figure, the two types of data sets have significant overlap in the middle

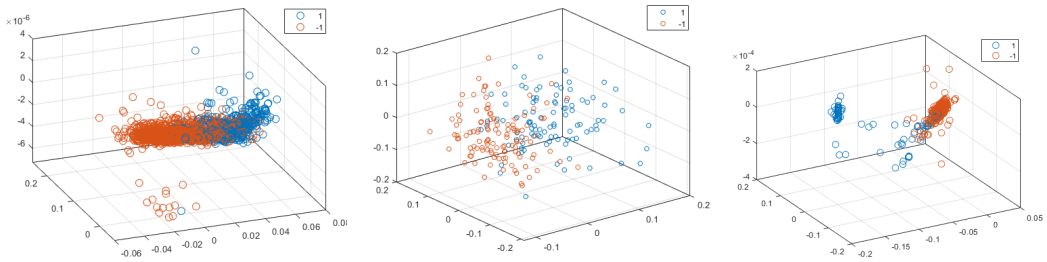
part for different parameter combinations, making it challenging to classify the data accurately. This overlap is also the primary reason for the low accuracy of the **a2a** data set in our previous classification experiments. Thus, to improve the classification accuracy of the model, we need to preprocess the data to make it easier to classify.

In contrast, the **heart** data set contains a relatively small amount of data, allowing us to observe the distribution of each data point clearly in two-dimensional space, as shown in Figure 3.11(b). Although there is a certain overlap between the positive and negative data, the classification experiment using the R-OPLS model still achieved an average classification accuracy of 85.94%.

In the w2a data set classification experiment, the prediction results reached high accuracy rates of 97% and 87% using classifier v and classifier W, respectively. The two-dimensional visualization of the w2a data in Figure 3.11(c) indicates that this is the most suitable data for classification among our three experimental data sets. The distributions of the two classes of data are scattered without much overlap, which corresponds to the high prediction accuracy achieved earlier. However, it is worth noting that the **w2a** data set is an unbalanced data set, as shown in Figure 3.11(c). The negative class has more data than the positive class, which can lead to highly misleading experimental results due to this class imbalance.

In addition to the two-dimensional visualization shown in Figure 20, we also utilized the *gscatter*3 [65] function, which is created by Salai Selvam V in MATLAB, to draw the scatter plots of the data sets in 3D space. Figure 3.12 displays a projection of the three datasets in a three-dimensional space. The three-dimensional visualization provides a more comprehensive representation of the data and allows us to explore the data distribution in a more intuitive way. By analyzing these visualizations, we can obtain a deeper insight into the distribution of the data and make

more informed decisions for our classification tasks.



(a) a2a Data Set (b) heart Data Set (c) w2a Data Set

Figure 3.12: The Visualization on 3-D Space

After analyzing the 3-D visualization of the a2a data set in Figure 3.12, we can conclude that it is an imbalanced data set. The vertical axis expansion, which cannot be represented in the 2-D visualization, reveals the data imbalance. In the 2-D visualization, there is a significant overlap between the two classes, making it challenging to determine if the data is imbalanced. As shown in Figure 3.12, the a2a data is distributed compactly in space, which poses a significant challenge for our classification experiments. Understanding the distribution of data is essential to overcome the challenge of classifying imbalanced datasets, and 3-D visualization enables us to gain deeper insights into the data sets.

It is noteworthy that the use of three-dimensional visualization allows us to observe the data from different perspectives, which can help us identify patterns that might not be visible in lower-dimensional spaces. These visualizations are crucial for us to determine the suitability of our datasets for classification and identify potential challenges in classifying them accurately. For instance, we can observe the distance between different data points and the relationship between different features. This

information can be used to identify potential outliers and evaluate the effectiveness of the feature selection process. Furthermore, these visualizations can also help us to validate the results obtained from the classification models and understand the reasons for misclassifications. Therefore, the combination of two-dimensional and three-dimensional visualizations is essential in the analysis of datasets and can lead to better-informed decisions in classification tasks. Therefore, through visualization, we can gain insight into the distribution of data in low-dimensional space. In addition, we need to be aware of the limitations of our data sets, such as class imbalance, to avoid misleading experimental results.

CHAPTER 4

A Novel Regularized OPLS Model for Multi-class Classification

4.1 R-OPLS Multi-Class Classification Model

Multi-class classification is a type of classification problem where we have more than two classes to predict. It is an extension of the binary classification problem, where we have only two classes to predict [18, 57]. One of the main differences between binary classification and multi-class classification is that they use different activation functions [57]. In binary classification, the *signum* function [1] is commonly used as the activation function for classification experiments, directly converting the prediction results into -1 and 1 [57]. In the other words, the threshold for this function is set to 0. If the predicted value is greater than zero, the activation function is considered a positive example with the label 1 [18]. Otherwise, the activation function is considered as a negative sample labeled -1 [57]. This approach is very effective for binary classification problems, such as spam email classification and fraud detection [54].

However, in multi-class classification problems, we need to predict more than two classes. While binary classification and multi-class classification share some similarities, such as the use of neural networks, they differ in the activation functions used to make predictions [57]. The *signum* function is effective for binary classification problems, while the $softmax$ function is used for multi-class classification problems [85, 57]. The $softmax$ function is a generalization of the logistic function [73], which can be used for multi-class classification problems. The output layer using $softmax$ has multiple units, equal to the number of classes in the data set [54]. Each unit

calculates the probability that the testing data points belong to a particular class [54, 85]. In this way, we can predict the probability of a sample belonging to each class and choose the class with the highest probability as the predicted class.

Alternatively, to solve multi-classification problems, we can use a technique that transforms them into multiple binary classification problems [54, 57]. In binary classification, we only need to learn a classifier between two classes of data. However, in multi-classification, we need to train multiple binary classifiers, and the number of classifiers depends on the method we choose. In general, to construct a c class discriminant in multi-class classification, we can combine multiple binary discriminant functions[18, 32]. Two common approaches to learning a classifier in multi-class classification are *One-vs-Rest* (OVR) and *One-vs-One* (OVO) [18, 32]. The OVR method needs to train c classifiers, while the OVO method needs to train $c(c-1)/2$ classifiers, where c is the number of classes in the data set and greater than 2 [32].

The OVR method is heuristic method for multi-classification by using binary classification algorithm that involves training c classifiers [32]. Each classifier is trained according to the rule, one class is positive and the rest are negative, to distinguish a class from other classes in the data set [18]. The multi-class data set is thus partitioned into multiple binary classification problems [54]. Train a binary classifier for each binary classification problem and use the most confident model to make predictions. During testing, if only one classifier predicts a positive class, the corresponding class label is used as the final classification result [57]. If multiple classifiers predict positive classes, the class with the highest confidence is selected as the final classification result [57].

The OVO method, on the other hand, requires training $c(c-1)/2$ classifiers. This method involves combining every two classes into a pair and treating each pair

as a binary classification task. Therefore, there are a total of $c(c-1)/2$ binary classification tasks in the OVO classifier [18]. During testing, test instances are fed into these binary classifiers, which then vote and calculate which class has the most results [32]. Both the OVR and OVO methods are heuristic methods for multi-classification using binary classification algorithms. The choice of method depends on the specific problem and the characteristics of the data set [18, 32].

4.2 Numerical Experiments

4.2.1 Data Information

In this section, the experimental results are based on three different data sets: **dna** [38], **usps** [34], and **protein** [16]. We provide a summary of the data information in Table 4.1, including the number of classes, dimensions, the number of training sets, and the number of testing sets. The experiments were conducted on a desktop computer equipped with an Inter(R) Core(TM) i7-4790 CPU and 32-GB RAM using MATLAB 2021b.

Data	Classes	Features	Training	Testing
dna	3	180	2000	1186
usps	10	256	7291	2007
protein	3	357	17766	6621

Table 4.1: Data Information

The **dna** data set, as reported in a study [38], comprises of 3186 splice junctions. Splice junctions represent the points on the DNA sequence where redundant DNA is eliminated during the formation of proteins in complex organisms [21, 38]. To analyze this data set, Ross King, a researcher at the University of Strathclyde, converted the initial 60 symbolic attributes into 180 binary features, which resulted in a StatLog

version of the data set [21, 38]. As a result, the data set is now represented by 180 binary attributes and requires classification into three classes, such as *ei*, *ie*, and *neither*. Here, *e* signifies exons, while *i* indicates introns [38]. The aim of analyzing this data set is to identify the correct class for each splice junction based on the available binary attributes. In other words, the task is to build a classification model that can accurately predict whether a splice junction belongs to the exon-intron (*ei*), intron-exon (*ie*), or neither (*neither*) category.

The United States Postal Service sponsored a research project on handwritten text recognition, and the second data set **usps** chosen for our analysis was derived from the project [34]. The raw data collection was conducted at the primary post office in Buffalo NY, between 1987 and 1988 [34]. The data set comprises of raw grayscale images of city, state, and zip codes, as well as bi-tonal images of alphabetic and numeric characters [34]. For our experiments, we focused on recognizing the numeric characters in the data set. Each location in the zip code has a true value that is matched with the number generated by the segmentation algorithm [34]. The training set is composed of randomly selected numbers from verified zip codes, with 7291 zip codes extracted from a total of 18468 for the experiment's training set [34]. The original test set consisted of 2711 digit images [34], from which we selected 2007 digit well-segmented images as the test set for our current experiments. Our R-OPLS model can accurately recognize numeric characters in postal codes based on available image data. By enabling this, we can facilitate the automation of postal code recognition and potentially increase the efficiency of the USPS mail processing system.

The third data set selected for multi-class classification is the **protein** data set from the Structural Classification of Proteins (SCOP) database [16]. The primary objective of the SCOP database is to offer the public access to a collection of protein

sequences, which have been classified based on their structure and function into a hierarchical arrangement [16]. The ultimate goal is to provide a thorough and detailed representation of the structural and evolutionary associations among all proteins with a known structure [16]. Our R-OPLS model can accurately identify the class of each protein sequence based on its structural and functional features. The features available in the data set can be used to classify proteins into different classes such as enzymes, receptors, and transporters, etc.

4.2.2 Multi-class Classification on Input Space

There are many other mature models and algorithms for multi-class classification such as K-Nearest-Neighbors(KNN) [40, 67], Random Forest [14, 17], and Decision Tree [4], we mentioned earlier. In this section, we chose the SVM, KNN, and Random Forest to classify the multi-class data sets on input space.

First of all, let's briefly introduce what Random Forest [4, 14, 17] is. The Random Forest is an extension of the well-known decision tree algorithm for regression and classification [4]. It creates a collection of classification trees by using a bootstrap sample of training data and random features of the trees [17]. The ensemble then aggregates to make a prediction [14]. Each decision tree in the forest judges and classifies new input samples, and each decision tree has its own classification results [4]. The final result of the Random Forest is based on the classification with the most votes [17]. The Random Forest consist of four steps [4]: 1) Random sampling and training decision trees; 2) Randomly choose a node splitting feature; 3) Repeat step 2 until no more splits; 4) Repeat step 1-3 to build a large number of decision trees to form a random forest. In our experiments, we used the $TreeBagger$ [14] function within MATLAB to build the random forest model for multi-class classification. The $TreeBagger$ grows each tree based on the $ClassificationTree$ [14] function, which

can take a randomly selected number of features for each decision split as an optional input parameter. The number of grown trees is a crucial parameter in $TreeBagger$, as the error decreases with an increase in the number of grown trees.

The evaluation of the performance of our multi-class classification model in the input space will be based on the use of confusion matrix [14, 47]. The confusion matrix is a widely used performance evaluation tool for classification problems in machine learning[4, 53]. It is a table that summarizes the classification results of a model by showing the number of correct and incorrect predictions for each class[4, 53]. Each column of confusion matrix represents the predicted class of the data point, while each row represents the actual class of the data point. The diagonal blocks of confusion matrix contain the true prediction for each class. Therefore, according to the confusion matrix, there are several important measures that can be defined such as accuracy, precision, and recall [4, 53].

$$\text{Accuracy} = \frac{\text{Total Number of True Prediction}}{\text{Total Number of Sample Size}};$$

$$\text{Precision} = \frac{\text{Number of True Prediction in Each Class}}{\text{Number of Total Prediction in Each Class}};$$

$$\text{Recall} = \frac{\text{Number of True Prediction in Each Class}}{\text{Actual Number of data in Each Class}};$$

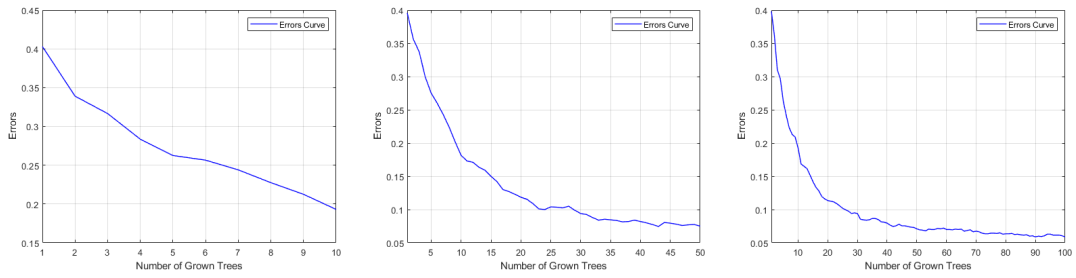
Specifically, in Figures 4.2, 4.4, and 4.6, we present the confusion matrices for the **dna**, **usps**, and **protein** datasets, respectively. These matrices provide a clear and concise visualization of the classification results, which will be used to calculate the accuracy, precision, and recall metrics. We will focus our analysis on accuracy, as it is a widely-used and effective indicator for evaluating the overall performance of the model.

Although all three indicators are useful in assessing model performance, we will focus on accuracy for simplicity and consistency in subsequent experiments. However, it is important to note that precision and recall can provide additional insights

into the model's performance, particularly when dealing with imbalanced datasets or scenarios where certain types of errors are more costly than others. Therefore, future work could explore the use of all three indicators and their trade-offs in different multi-class classification scenarios.

dna Data Set

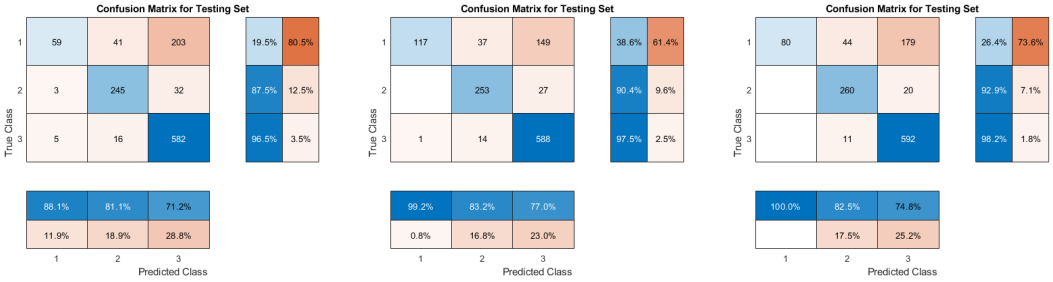
We conducted experiments on the **dna** data set to evaluate the performance of the random forest model. Specifically, we experimented with three different values for the number of trees to be grown: 10, 50, and 100. To determine the optimal number of grown trees for the highest accuracy, we analyzed the error rate for each value by plotting it against the number of grown trees.



(a) Grown Trees=10 (b) Grown Trees=50 (c) Grown Trees=100

Figure 4.1: Error Curves on **dna** Data Set

The resulting error curves are shown in Figure 4.1. The plots demonstrate that when the number of grown trees is less than 40, there is a significant change in the error curve. However, when the number of grown trees exceeds 50, the error curve becomes relatively flat and converges. Thus, we concluded that approximately 50 trees is an appropriate choice for the number of grown trees, and further experiments will be conducted to validate this hypothesis.



(a) Grwon Trees=10 (b) Grwon Trees=50 (c) Grwon Trees=100

Figure 4.2: Confusion Matrices on **dna** Data Set

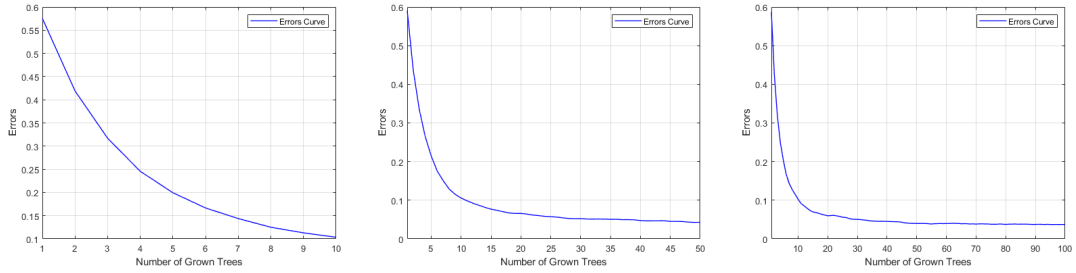
To evaluate the performance of the random forest model, we used confusion matrices and three indicators, namely accuracy, precision, and recall. Figures 4.2(a) to 4.2(c) depict the confusion matrices of the random forest algorithm with three different numbers of grown trees. By analyzing these confusion matrices, we observed that the precision of class 1 in the random forest model increases as the number of grown trees increases. However, we also observed that a significant amount of data belonging to class 1 was misclassified as class 3 in each experiment, resulting in relatively low precision of class 3 and very low recall of class 1.

Furthermore, we found that the recall of class 1 is highest when the number of grown trees is 50, but it is only 38.6%. Therefore, when the number of grown trees is equal to 50, the overall accuracy rate is the highest at 81.87% for the multi-class classification experiment based on random forest for the **dna** data. It is worth noting that although the precision of class 1 reaches 100% when the number of grown trees is 100, the recall rate of class 1 is still low, and a large amount of data is still wrongly classified into class 3. Thus, selecting the appropriate number of grown trees is crucial to achieving high accuracy rates in multi-class classification experiments. Overall, our findings suggest that the random forest model is effective for multi-class classification tasks, but careful selection of hyperparameters is necessary to achieve

optimal performance.

usps Data Set

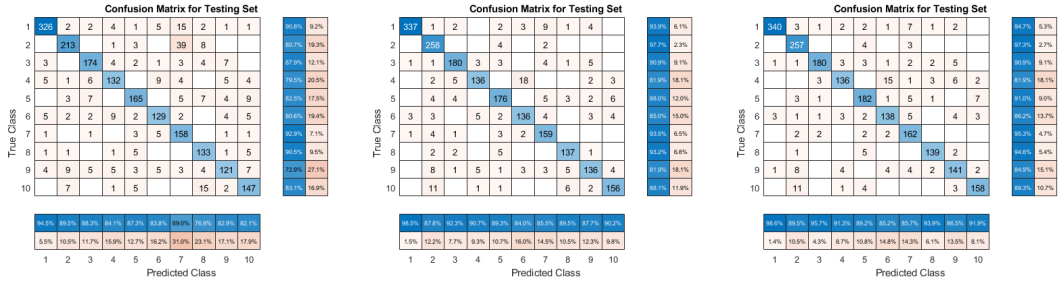
The selection of an optimal number of trees is crucial for achieving accurate predictions using a random forest algorithm. In this regard, we conducted experiments with different numbers of trees and plotted the corresponding error curves for the **usps** dataset. Initially, we set the number of trees to 10 and observed the error curve, which is shown in panel (a) of Figure 4.3. As we increased the number of trees, the error rate continued to decrease, indicating that the model had not yet converged. Thus, we increased the number of trees to 50 and plotted the error curve, as shown in panel (b) of Figure 4.3. We noticed that the error curve changed slowly, but it did not completely stabilize even after growing 20 trees. Therefore, we further increased the number of trees to 100 and plotted the corresponding error curve, as shown in panel (c) of Figure 4.3. After analyzing the error curves in panel (c), we observed that the error rate for the first 50 trees dropped significantly and then stabilized, indicating that the model with 50 trees fits the dataset well. However, we still need to confirm this conjecture through further experiments.



(a) Grown Trees=10 (b) Grown Trees=50 (c) Grown Trees=100

Figure 4.3: Error Curves with Grown Trees on **usps** Data

The random forest classification results and confusion matrices on the **usps** data set are shown in Figures 4.4(a)-4.4(c). The test set for each category had a relatively balanced amount of data, with about 200 data points in each category, except for category 1, which had over 300 data points. Furthermore, we observed that some data points in each class were misclassified as other classes. In particular, the precision of class 7 increased significantly with an increase in the number of grown trees. When the number of grown trees was 10, the precision of class 7 was only 69%. However, as the number of grown trees increased to 50 and 100, the precision of class 7 reached more than 85%. Similarly, as the number of grown trees increased, the precision and recall of other classes also correspondingly increased.



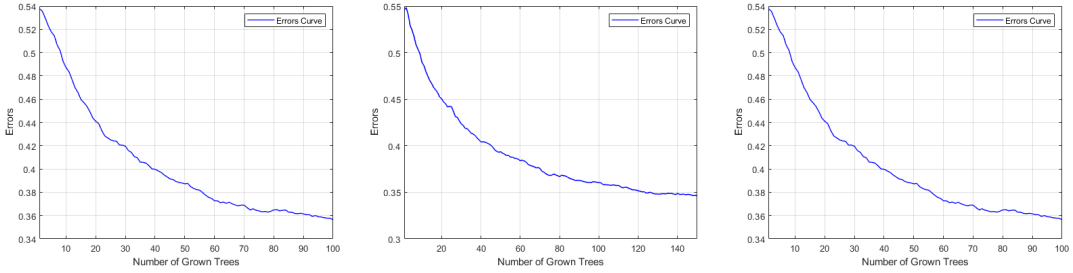
(a) Grwon Trees=10 (b) Grwon Trees=50 (c) Grwon Trees=100

Figure 4.4: Confusion Matrices on **usps** Data Set

protein Data Set

Similar to the previous two sets of experimental data, we conducted a random forest experiment and investigated the error graphs for different numbers of grown trees for the **protein** data set. We planted the grown trees ranging from 50 to 300 in increments of 50 and examined the error curves for 100, 150, and 200 trees, as depicted in Figure 4.10. The error curve indicates that the accuracy of the algorithm increases with the number of grown trees until a threshold is reached, beyond which

the performance saturates or even deteriorates due to overfitting. In this case, the error curves continued to decrease as the number of grown trees was less than 200, indicating that more trees were necessary to enhance performance. However, the curve started to flatten when the number of grown trees grew beyond 200, indicating that further increasing the number of grown trees did not significantly improve performance. Thus, we concluded that 200 is the optimal number of grown trees for the Random Forest algorithm with the **protein** data set. These results can serve as a reference for future experiments or practical applications of protein data classification using the random forest algorithm.

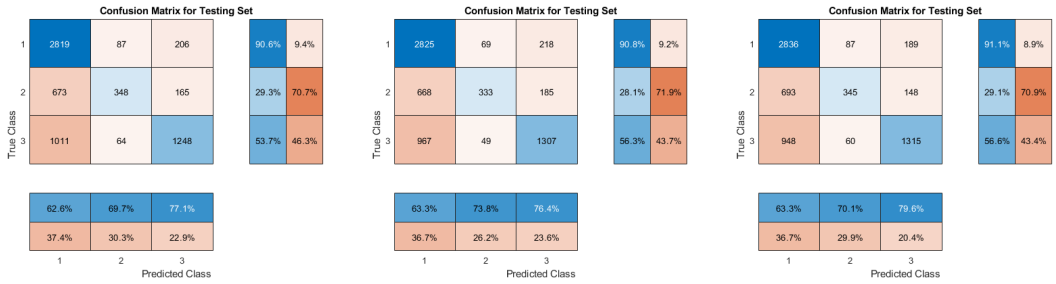


(a) Grown Trees=100 (b) Grown Trees=150 (c) Grown Trees=200

Figure 4.5: Error Curves with Grown Trees on **protein** Data

To evaluate the generalization performance of the random forest method, we plotted the confusion matrix for different numbers of grown trees. Figure 4.6 presents the performance of the random forest method when the number of grown trees was 100, 150, and 200, respectively. The results showed that growing 200 trees led to higher recall and precision per class of data. Recall measures the proportion of true positive instances that are correctly classified, while precision measures the proportion of positive predictions that are correct. The confusion matrix revealed that class 1 had the highest recall, indicating that most instances belonging to class 1

were correctly classified. On the other hand, class 2 had the lowest recall, indicating that the algorithm had difficulty distinguishing class 2 from the others. The accuracy value for each class reflected the overall accuracy of the algorithm for that class, with higher values indicating fewer false positives.

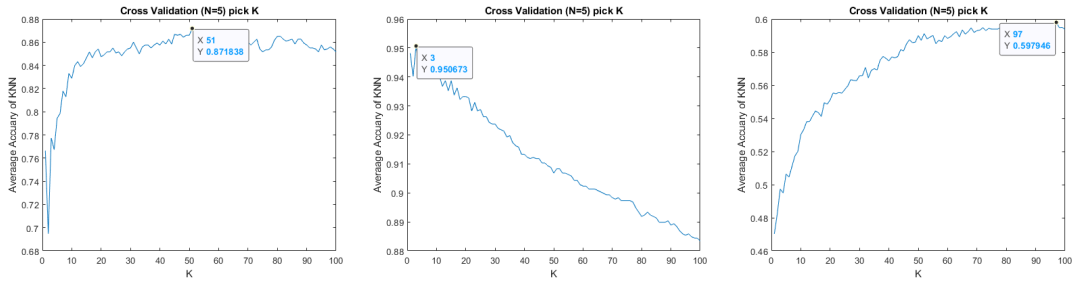


(a) Grwon Trees=100 (b) Grwon Trees=150 (c) Grwon Trees=200

Figure 4.6: Confusion Matrices of Random Forests on **protein** Data

In addition to the Random Forest method, we also employed the KNN [36] algorithm to classify three multi-class data sets: **dna**, **usps**, and **protein**. The KNN algorithm is a straightforward yet effective classification technique that assigns a new instance to the class that is most common among its K nearest neighbors in the training set. However, determining the optimal value of K for a given data set is critical for achieving good generalization performance [71]. We used the 5-folder cross-validation [60] method to evaluate the performance of the KNN algorithm. This method partitions the data into five equal subsets, trains the model on four of them, and evaluates it on the remaining subset [1, 36]. This process is repeated five times, with each subset serving as the testing set once. The results are then averaged to obtain an estimate of the algorithm's performance on unseen data [36, 67, 71].

Figure 4.7 shows the optimal value of K for each data set. For the **dna** data set, the optimal value of K is 51, indicating that the algorithm considers the 51



(a) **dna** Data Set (b) **usps** Data Set (c) **protein** Data Set

Figure 4.7: Accuracy of KNN on Input Space

nearest neighbors of a new instance to make a classification decision. For the **usps** data set, the optimal value of K is 3, suggesting that the algorithm only considers the three nearest neighbors in this case. Finally, for the **protein** data set, the optimal value of K is 97, indicating that the algorithm needs to look at a larger number of neighbors to achieve optimal performance.

It is worth noting that the choice of K affects the bias-variance trade-off of the algorithm. Smaller values of K result in higher variance and lower bias, while larger values of K lead to lower variance and higher bias. Selecting the optimal value of K is crucial for achieving good generalization performance and avoiding overfitting or underfitting the data.

Table 4.2 shows the classification accuracy results of three multi-class data sets, namely **dna**, **usps**, and **protein**, using three different classifiers in the input space. The classifiers used were KNN, Random Forest, and Least Squares.

Data Set	KNN	Random Forest	Least Squares
dna	87.18%	81.87%	88.87%
usps	95.07%	90.23%	87.29%
protein	59.79%	67.91%	63.30%

Table 4.2: Summary of Classification Accuracy on Input Space

In general, the Table 4.2 suggests that KNN and Least Squares performed better than Random Forest in terms of classification accuracy. Among the three classifiers, Least Squares achieved the highest classification accuracy on the **dna** data set, KNN achieved the highest accuracy on the **usps** data set, and Random forest has the best performance on protein data set.

For the **dna** data set, least squares method achieved an accuracy of 88.87%, which is higher than the accuracy achieved by Random Forest (81.87%) and KNN (87.18%). Similarly, for the **usps** data set, KNN achieved the highest accuracy of 95.07%, followed by Random Forest with an accuracy of 90.23%, while Least Squares had the lowest accuracy of 87.29%. For the **protein** data set, Random Forest had the highest accuracy of 67.91%, followed by Least Squares with an accuracy of 63.30%, while KNN had the lowest accuracy of 59.79%. Overall, the results suggest that the choice of classifier is data-dependent, and different classifiers may perform differently on different data sets.

Based on the aforementioned experiments, it can be observed that KNN, Random Forest, and Least Squares methods can only perform classification in the original data space without the possibility of dimensionality reduction. However, our newly developed R-OPLS model can effectively reduce the dimensionality of data while preserving high classification accuracy. By projecting the data onto a low-dimensional subspace using the projection matrix P of the R-OPLS model, classification experiments can be conducted with higher efficiency. This can greatly improve the overall experimental efficiency. In the next section, we will discuss the classification experiments conducted using R-OPLS in the projected subspace for the aforementioned three data sets.

4.2.3 4.2.3 Multi-class Classification on Projected Subspace

4.2.3.1 $Softmax$ Function

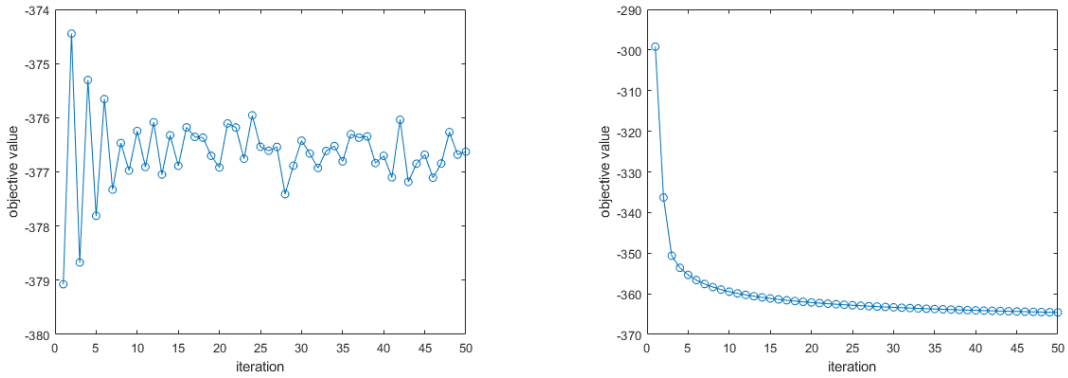
In this section, we utilized the one-hot representation of class labels to represent the indicator matrix $Y \in \{0,1\}^{c\times n}$ and the centered matrix $\hat{Y} = YH \in \mathbb{R}^{c\times n}$, where $H = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T \in \mathbb{R}^{n\times n}$ is the centering matrix. By applying mathematical deduction of R-OPLS, we obtained the prediction result as a $\mathbb{R}^{c\times n}$ matrix. Subsequently, the $softmax$ activation function was applied to calculate the probability of the testing data points belonging to each class. Consequently, we obtained a $\mathbb{R}^{c\times n}$ matrix, where each column corresponded to the prediction information for each testing data point, and each row contained the probability that the testing data belonged to each class.

To obtain the predicted labels for the testing data set, we set the maximum value of each column in the $softmax$ result to 1 and the remaining values to 0. We achieved this by employing the $vec2ind$ function available in MATLAB. In summary, the process involved the transformation of the probability matrix to an index matrix, where the index represented the predicted label for each testing data point. By using the one-hot representation of the class labels and applying the $softmax$ activation function, we were able to generate the predicted labels for the testing data set.

dna Data Set

For the alternating iteration experiment on the **dna** data set, we followed the same initial parameter settings as in the binary classification experiment. The parameter C was selected as 0.01, 0.1, 1, and 10, and the parameter λ was set to 0.1, 0.5, 0.7, and 0.9. We employed the *quadprog* solver and the projected gradient descent (PGD) method for alternating iteration experiments. It was observed that

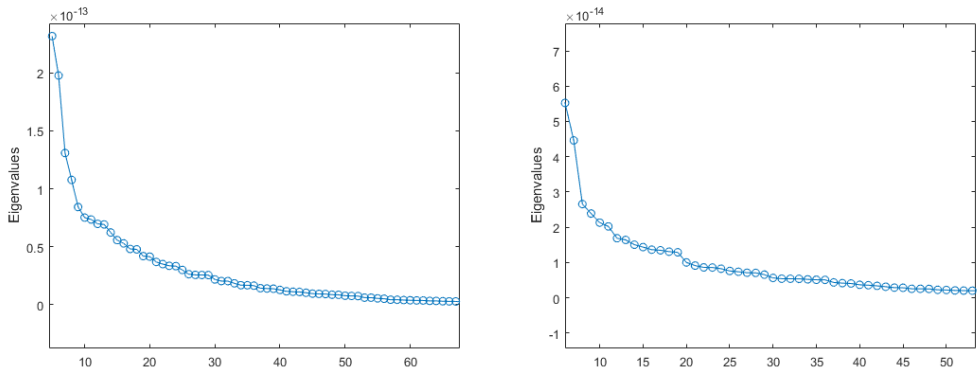
the convergence of the objective function was significantly affected by the parameter λ. After conducting several experiments, we found that the optimal value for λ was 0.5. Additionally, the optimal value for C was determined to be between 1 and 10. To determine the best parameter combination for the dna data when using the $softmax$ function, we performed 5-fold cross-validation. The optimal parameter combination was found to be $C = 1.5$ and $\lambda = 0.5$. Figure 4.8 illustrates the curves in the objective function during the alternate iteration experiments when the *quadprog* solver and PGD were used, respectively.



(a) *quadprog* (b) PGD

Figure 4.8: Iteration Plots on **dna** Data

To determine the appropriate projection dimension k, we examined the eigenvalue curve of the **dna** dataset obtained using the *quadprog* solver and projected gradient descent method, as shown in Figure 4.9. We observed that the eigenvalue curve starts to flatten after 60 when using both methods, suggesting that we can effectively reduce the dimensionality of the data while preserving its class-discriminating information.



(a) *quadprog* (b) PGD

Figure 4.9: The Plots of Eigenvalues on **dna** Data

To evaluate the effectiveness of R-OPLS model for multi-class classification tasks on the **dna** data set, we conducted a classification experiment by randomly selecting 80% of the data as the training set and setting the remaining 20% as the test set. We varied the value of k from 10 to 60 and repeated the experiment 10 times at each value of k, recording the results and taking the average.

Solver \ k	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
quadprog	$89.59 \pm 1.79\%$	$89.50 \pm 0.78\%$	$89.57 \pm 1.65\%$	$89.29 \pm 2.93\%$	$89.84 \pm 2.17\%$
PGD	$88.86 \pm 2.05\%$	$89.23 \pm 1.84\%$	$89.89 \pm 1.49\%$	$89.12 \pm 1.48\%$	$89.28 \pm 1.63\%$

Table 4.3: Accuracy with Selection of k on **dna** Data

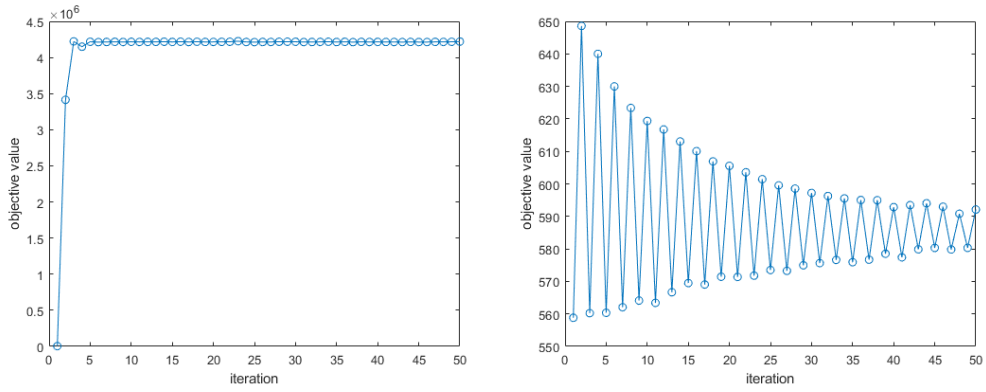
These results demonstrate the effectiveness of our R-OPLS algorithm for reducing the dimensionality of the **dna** data set while preserving its class-discriminating information, leading to high classification accuracy.

usps Data Set

The **usps** data set under consideration comprises of raw data descriptions consisting of both gray-scale images and bi-tonal images of alphabetic and numeric characters, pertaining to city, state, and zip code information[34]. The scope of our experiments focuses on recognizing the numeric characters within the zip code images. To achieve this, [34] employ a segmentation algorithm that maps each location within the zip code image to a corresponding numerical output value. The training set used in our experiments comprises randomly selected numbers from verified zip codes, with a total of 7291 zip codes extracted from a pool of 18468 zip codes[34]. The original test set consists of 2711 digital images[34]. However, [34] have hand-picked 2007 of these images, which exhibit well-segmented numeric characters, to use as the testing set for the experiments. To ensure that the performance of our model is not affected by poorly segmented characters or other artifacts in the data, we redistribute the training and testing data. We integrate all given training and test data and randomly select 80% of them as the training set, and the remaining 20% as the test set.

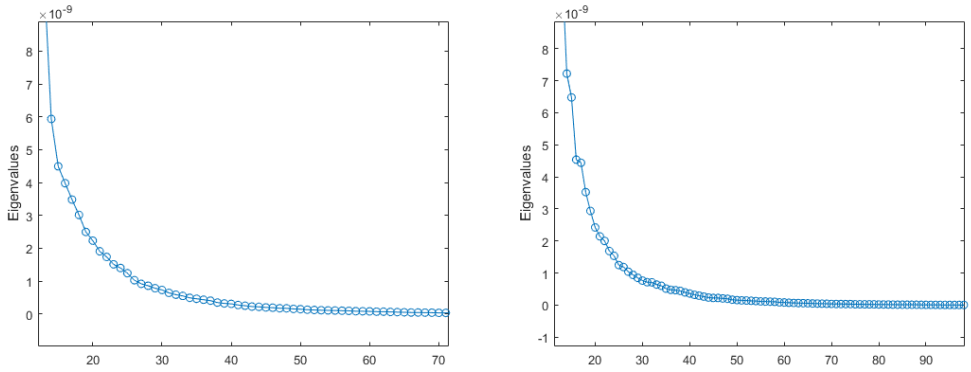
To determine the optimal parameter combination for the **usps** data set, we conduct alternating iterative experiments combined with five-fold cross-validation, and find that the optimal parameter combination is C=1 and λ=0.5. We use the *quadprog* solver and the projected gradient descent method to conduct alternate iterative experiments. Figure 4.10 shows the change curve of the function value when the *quadprog* solver and the projected gradient descent method are used respectively under the optimal parameter combination.

To gain insight into the correlation of features in the **usps** data set, we analyze the variation of eigenvalue curves and select the most relevant features. Figure 4.11 shows the eigenvalue plot, which indicates that the eigenvalue curve flattens after 30 iterations in the *quadprog* solver and PGD.



(a) *quadprog* (b) PGD

Figure 4.10: Iteration Plots on **usps** Data



(a) *quadprog* Solver (b) Projected Gradient Descent

Figure 4.11: The Plots of Eigenvalues on **usps** Data

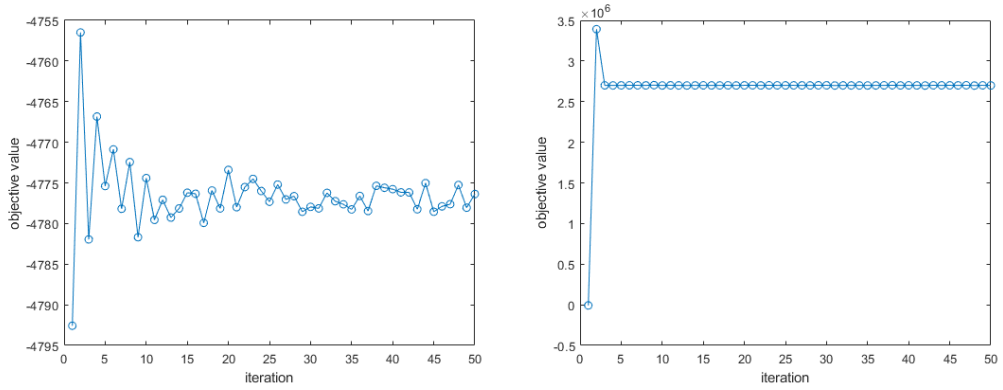
To evaluate the performance of our R-OPLS algorithm on the **usps** data set, we conduct classification experiments by varying the projection dimension k between 10 and 60. For each value of k, we repeat the experiment 10 times and take the average value. The experimental results are reported in Table 4.4. Our results demonstrate high classification accuracy on the projected subspace using R-OPLS for multi-class classification tasks on the **usps** data set.

protein Data Set

Solver \ k	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
quadprog	$90.63 \pm 0.82\%$	$90.07 \pm 0.74\%$	$90.29 \pm 0.78\%$	$89.97 \pm 0.59\%$	$90.37 \pm 0.71\%$
PGD	$90.39 \pm 0.42\%$	$90.23 \pm 1.01\%$	$90.08 \pm 0.46\%$	$90.32 \pm 0.33\%$	$90.06 \pm 0.75\%$

Table 4.4: Accuracy with Selection of k on **usps** Data

Our experiment with the **protein** data set involved a redistribution of the training and testing data, where all the given data was combined, and 80% of it was randomly selected as the training set while the remaining 20% was set aside as the test set. We conducted the alternating iteration experiment with the *quadprog* solver and projected gradient descent method, while utilizing the 5-fold cross-validation technique to determine the optimal parameter combination for the **protein** data set. Through this analysis, we identified the optimal values of $C = 1$ and $\lambda = 0.5$, as shown in the convergence plots of the objective function values in Figure 4.12.

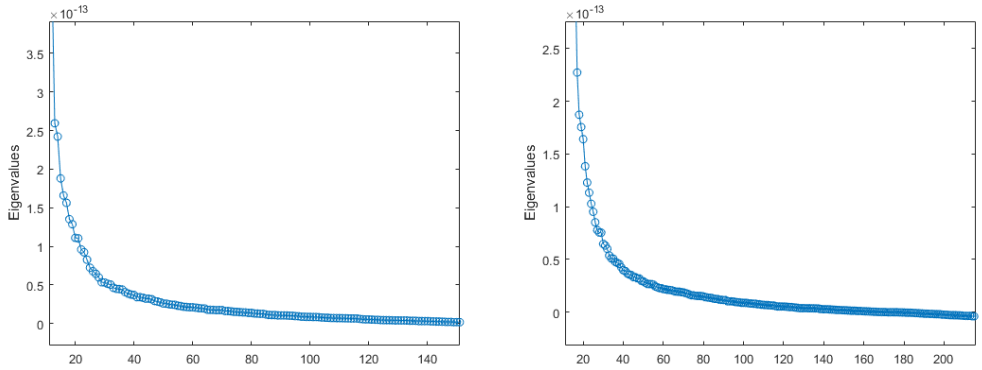


(a) *quadprog* (b) PGD

Figure 4.12: Iteration Plots on **protein** Data

Furthermore, we conducted eigenvalue analysis to gain a deeper understanding of the features present in the **protein** dataset. The eigenvalue plots in Figure 4.13

showed that the eigenvalue curves flattened after 60 iterations for both the *quadprog* solver and projected gradient descent method.



(a) *quadprog* Solver (b) Projected Gradient Descent

Figure 4.13: The Plots of Eigenvalues on **protein** Data

Based on these results, we performed a series of classification experiments for different values of k, ranging from 10 to 60. We repeated the experiments 10 times for each value of k and recorded the average classification accuracy. The results of these experiments are presented in Table 4.5.

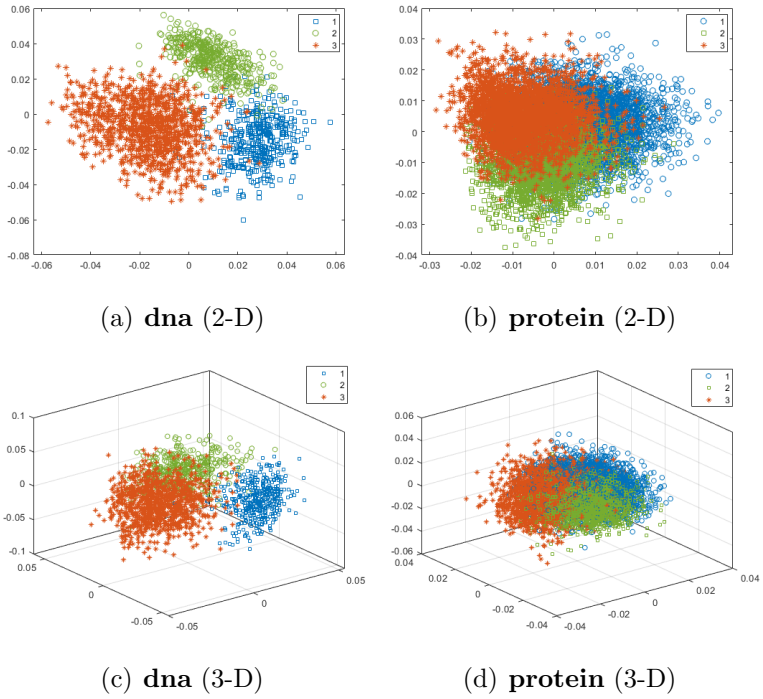
Solver \ k	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
quadprog	$65.81 \pm 1.14\%$	$65.93 \pm 0.59\%$	$65.98 \pm 0.60\%$	$66.32 \pm 1.25\%$	$65.87 \pm 0.87\%$
PGD	$65.32 \pm 1.39\%$	$65.75 \pm 0.98\%$	$65.71 \pm 0.92\%$	$66.14 \pm 1.35\%$	$65.59 \pm 0.58\%$

Table 4.5: Accuracy with Selection of k on **protein** Data

By analyzing the results in Table 4.5, we observed that the classification accuracy of the **protein** data set improved as we increased the value of k. The optimal projection space for this dataset was found to be $k = 60$, which provided the most discriminative features for accurate classification. When using the *quadprog* solver,

the classification accuracy achieved with $k = 60$ was 63.80% on the test set, which is the highest among all the tested projection spaces.

In order to gain a better understanding of the distribution of data in the **dna** and **protein** datasets, we performed dimensionality reduction to visualize the data in 2D and 3D space, as shown in Figure 4.14. By projecting high-dimensional data into a low-dimensional space, we can more easily perceive the underlying patterns and structures of the data, which can provide valuable insights for future analysis and interpretation.



(a) **dna** (2-D) (b) **protein** (2-D)

(c) **dna** (3-D) (d) **protein** (3-D)

Figure 4.14: Visualization

The 2D and 3D visualizations of the **dna** and **protein** data sets allow us to observe the distribution and clustering of the data points in a more intuitive manner. In the 2D visualizations, we can see the relationships between the different classes

and how they are distributed throughout the space. In the 3D visualizations, we can gain a more comprehensive understanding of the relationships between the data points and the patterns that emerge from the clustering. It should be noted that the **usps** data set contains a total of 10 classes, which can result in a cluttered and unclear scatter plot. Therefore, we have chosen not to include a spatial scatter plot of the **usps** data set here.

In this section, we present the results of our multi-class classification experiments using the R-OPLS model on three distinct datasets. Our experiments illustrate the R-OPLS model's effectiveness in reducing dimensionality while achieving high classification accuracy in the projected subspace.

We observed that the R-OPLS model generates two classifiers simultaneously, but we only demonstrated the R-OPLS classifier due to the SVM classifier's design for binary classification tasks. To address this limitation, we will introduce two methods, One-vs-Rest (OVR) and One-vs-One (OVO), to the R-OPLS model in next section. These methods transform multi-class classification problems into multiple binary classification problems, allowing us to use the R-OPLS model's two classifiers for classification prediction.

4.2.3.2 One-vs-Rest (OVR)

The OVR classifier is a popular approach for multi-class classification problems that utilizes binary classification algorithms. The method partitions the multi-class data set into multiple binary classification problems based on the rule of one class is positive, the rest are negative [32]. In this strategy, c binomial classifiers are trained, where c represents the number of classes in the data set. During testing, each classifier makes a prediction, and if only one classifier predicts a positive class, then the corresponding class label is assigned as the final classification result [57].

However, if multiple classifiers predict positive classes, the class with the highest confidence score is selected as the final classification result [57]. This approach can be effective for datasets with multiple classes and can provide accurate classification results.

Consider the example of the **dna** data set with 3 classes: C_1, C_2, and C_3. In the OVR method, we need to create 3 classifiers, namely f_1, f_2, and f_3. During the testing process, the classification process is as follows:

$+$	$-$		$classifiers$		$results$
$\boxed{C_1}$	$\boxed{C_2\ C_3}$	$\Rightarrow$	f_1	$\rightarrow$	$-$
$\boxed{C_2}$	$\boxed{C_1\ C_3}$	$\Rightarrow$	f_2	$\rightarrow$	$-$
$\boxed{C_3}$	$\boxed{C_1\ C_2}$	$\Rightarrow$	f_3	$\rightarrow$	$+$

The example above demonstrates how the OVR classifier works. During the testing process, we classify a test instance by applying all three classifiers to it. If only one classifier predicts a positive class, then we use the corresponding class label as the final classification result. For instance, if only f_3 predicts a positive class outcome and the other classifiers predict negative class outcomes, we predict the class label C_3 for that test instance. If more than one classifier predicts a positive class, we choose the class with the highest confidence level as the final classification result. In other words, we select the class label associated with the classifier that assigns the highest probability to the test instance belonging to that class.

dna Data Set

The **dna** data set is a challenging classification problem consisting of three distinct classes: *ei*, *ie*, and *neither*. To address this problem, we employed the One-vs-Rest (OVR) technique with the R-OPLS model to construct three binary

Algorithm 4.1 One-vs-Rest Multi-classification

Require: Training set X, consisting of n examples with d features and c possible class labels

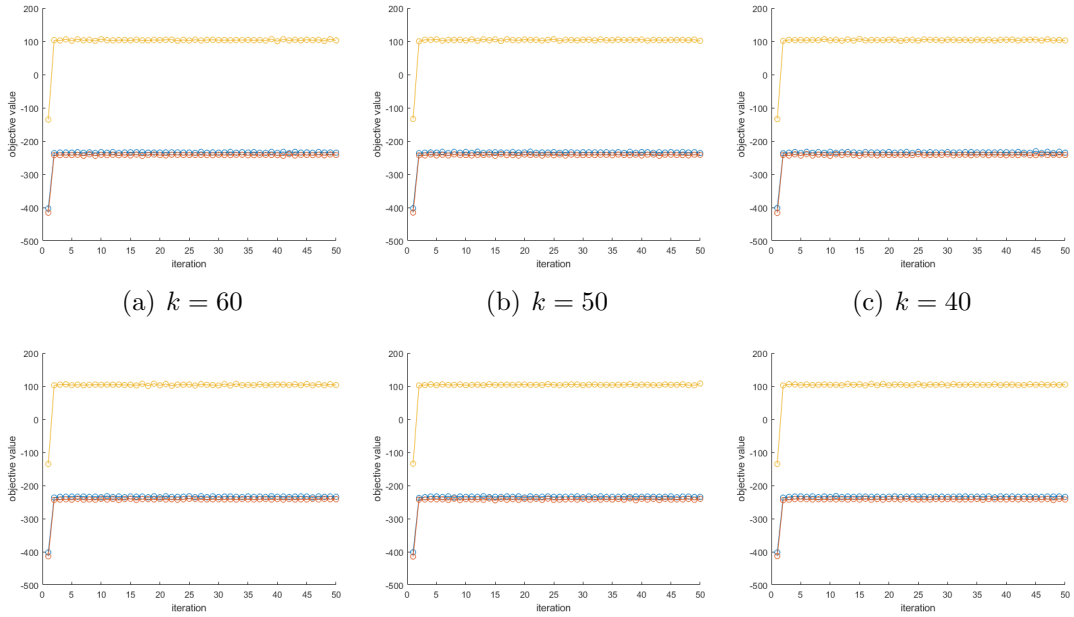
Ensure: Predicted class label for a new input vector x'

1: **for** $i = 1$ to c **do**
2: Construct a new binary training set X_i by relabeling the examples with class label i as positive and all other class labels as negative
3: Train a binary classifier $f_i(x)$ on X_i
4: **end for**
5: Compute the output of each binary classifier $f_i(x')$ for the new input vector x'
6: **Return** the class label with the highest output as the predicted label

classifiers. Each classifier was trained to distinguish between one positive class and two negative classes. By breaking down the classification task into separate binary problems, we were able to effectively handle the complex data and achieve accurate results.

In previous experiments, we analyzed the parameters C and λ and found their optimal values to be 1.5 and 0.5, respectively. We also observed that the iteration curve remained convergent across different values of k when these parameters were employed. Identifying the optimal values of parameters can improve the performance and accuracy of models. Moreover, the convergence of the iteration curve suggests that our models are robust and can generalize well to new data.The convergence of the iterative curves across different values of k is also demonstrated in the Figure 4.15. The results indicate that the chosen values of $C = 1$ and $\lambda = 0.5$ led to convergent iteration curves in each binary classification experiment. This suggests

that the models were effective in handling the complexity of the **dna** data set and were able to achieve accurate results.



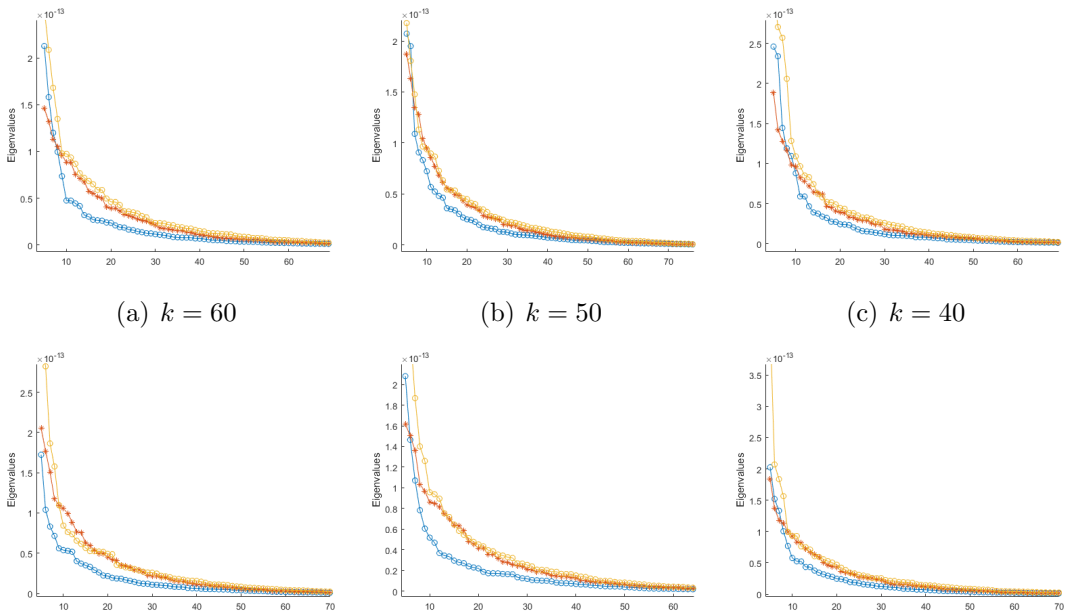
(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

(d) $k = 30$ (e) $k = 35$ (f) $k = 20$

Figure 4.15: OVR Method Iteration plots on **dna** Data

These findings have important implications for the application of machine learning techniques to the **dna** data set. By identifying the optimal values for C and λ, we can improve the performance and accuracy of our models. Furthermore, the convergence of the iteration curve across different values of k provides evidence of the robustness and stability of our models, and suggests that they are able to generalize well to new data. Overall, these results highlight the importance of parameter selection and model validation in machine learning. By conducting rigorous experiments and carefully analyzing our results, we can ensure that our models are accurate, reliable, and effective in real-world applications.

We conducted experiments on the **dna** data set by varying the value of k, and generated eigenvalue plots as shown in Figure 4.16. In each plot, we observed that the curves began to converge at around $k = 60$. This indicates that the effective number of features for each model is approximately 60. Therefore, in our prediction experiments, we utilized a dimensionality reduction space with a value range around $k = 60$. The eigenvalue plots provide valuable insights into the behavior of our models at different values of k. By analyzing these plots, we were able to determine the optimal value for k and identify the most important features for each model. This allowed us to refine our approach and improve the accuracy of our predictions. Furthermore, these results demonstrate the importance of dimensionality reduction in machine learning. By reducing the number of features in our models, we were able to improve their efficiency and accuracy, and obtain more meaningful insights from our data.



(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

(d) $k = 30$ (e) $k = 35$ (f) $k = 20$

Figure 4.16: eigenvalue plots on **dna** Data

As previously mentioned, our novel R-OPLS model allows us to obtain two classifiers: the R-OPLS classifier and the SVM classifier, simultaneously. To evaluate the performance of these classifiers, we conducted a classification experiment with varying values of k, ranging from 60 to 10. The results of this experiment are presented in Table 4.6, and indicate the accuracy achieved by each classifier.

Model \ **Data Selection**		k=60	k=50	k=40	k=30	k=20	k=10
dna	R-OPLS	88.87%	88.87%	88.87%	88.87%	88.87%	88.87%
	SVM	88.95%	89.12%	89.21%	89.46%	89.21%	88.87%

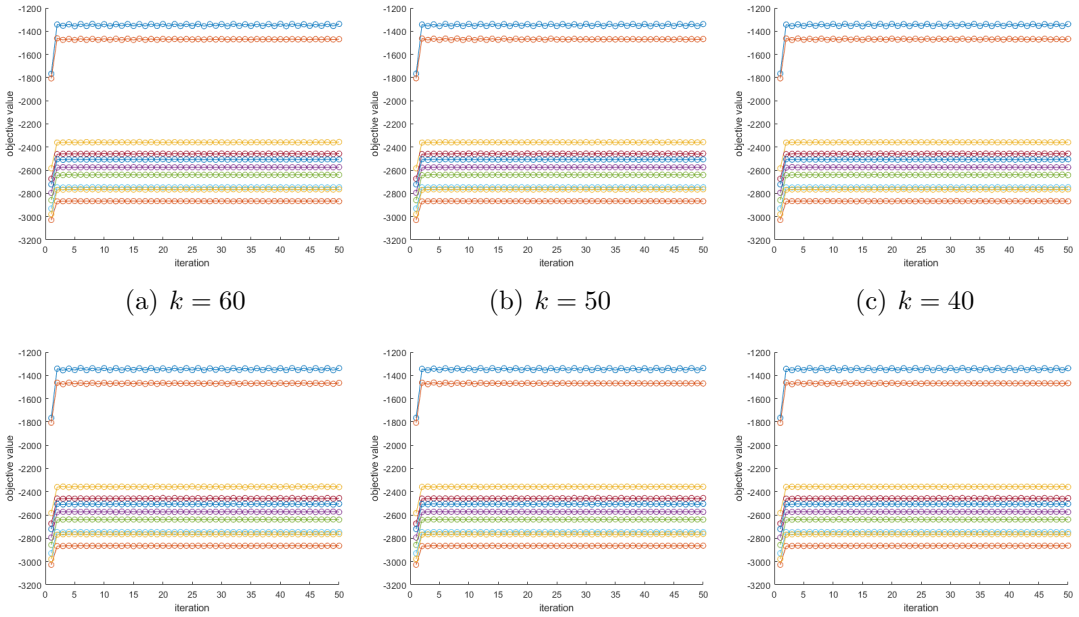
Table 4.6: Accuracy of OVR Multi-Class Classification for **dna** Data Set

The accuracy results show that both the R-OPLS-related classifier and the SVM-related classifier achieved high levels of accuracy across all values of k. These findings suggest that our R-OPLS model is an effective and reliable approach for classifying the **dna** data set. By simultaneously obtaining two classifiers, we can obtain more accurate and robust results, and gain deeper insights into the underlying patterns and relationships in the data. Additionally, the high levels of accuracy achieved by our models highlight the potential of machine learning techniques for solving complex real-world problems.

usps Data Set

The **usps** data set is a well-known benchmark data set that consists of handwritten digit images from $0-9$. The objective of the classification task is to predict the correct digit label for each image. However, since there are 10 classes in this data set, we cannot use a binary classifier directly for classification. To address this issue, we use the OVR strategy, which involves training 10 separate binary classifiers using

the R-OPLS model. Each binary classifier is trained to distinguish between one digit as the positive class and all other digits as the negative class. This approach effectively decomposes the multi-class classification problem into 10 binary classification problems.



(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

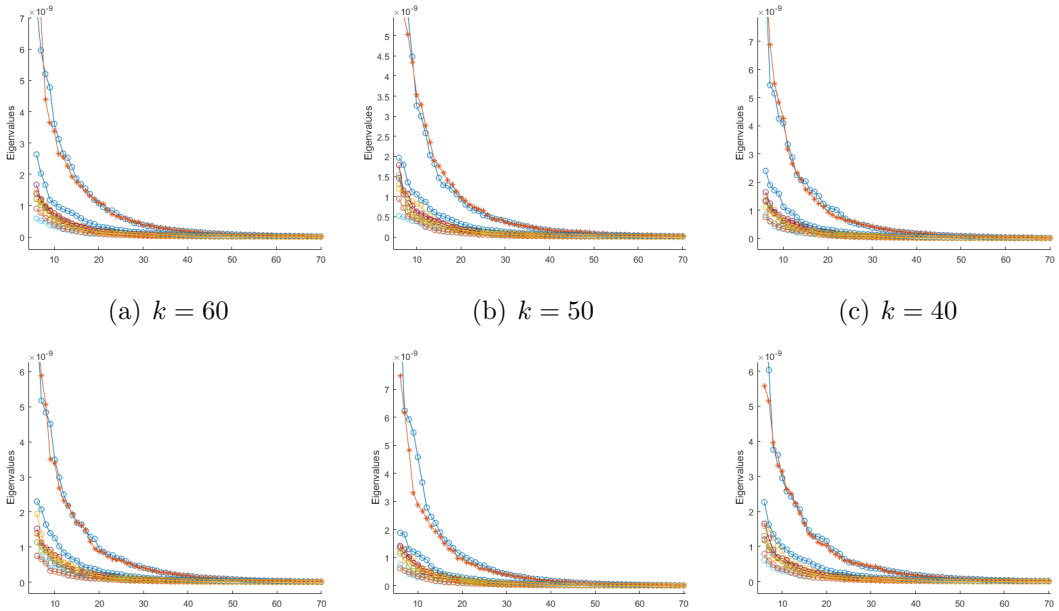
(d) $k = 30$ (e) $k = 20$ (f) $k = 10$

Figure 4.17: OVR Method Iteration plots on **usps** Data

During the training process, we apply several pre-processing techniques to the **usps** data set, including splitting the data into training and testing sets and standardizing the data to ensure that all features are on the same scale. The R-OPLS algorithm is then applied to each binary classification problem to learn the optimal subspace that separates the positive class from the negative class. To ensure that the trained models generalize well, we evaluate their performance on the held-out testing set. The testing data is projected onto the subspace found during training, and each binary classifier is used to predict the class label of each observation. The over-

all classification accuracy is then calculated as the percentage of correctly classified observations.

We monitor the convergence of the R-OPLS algorithm by plotting the objective function value against the iteration number. The objective function measures the separation between the positive and negative classes in the subspace. The Figure 4.17 objective function value decreases as the algorithm converges, indicating that the classes are becoming more separated. After the convergence analysis, we can also plot the eigenvalues of the subspace to see how many eigenvalues are needed to capture the majority of the variance in the data as shown in Figure 4.18. This analysis can help us choose the most informative features for classification.



(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

(d) $k = 30$ (e) $k = 20$ (f) $k = 10$

Figure 4.18: eigenvalue plots on **usps** Data

Once the optimal subspace is chosen, we can train the one-vs-rest classifiers using the R-OPLS subspace and evaluate the classification accuracy on the testing

set. The classification accuracy can be calculated as the percentage of observations that are correctly classified as shown in Table 4.7. Overall, the R-OPLS with OVR approach can provide a powerful and efficient way to classify high-dimensional data into multiple categories. By analyzing the convergence and eigenvalue plots, we can choose the optimal subspace and features for classification, resulting in high classification accuracy.

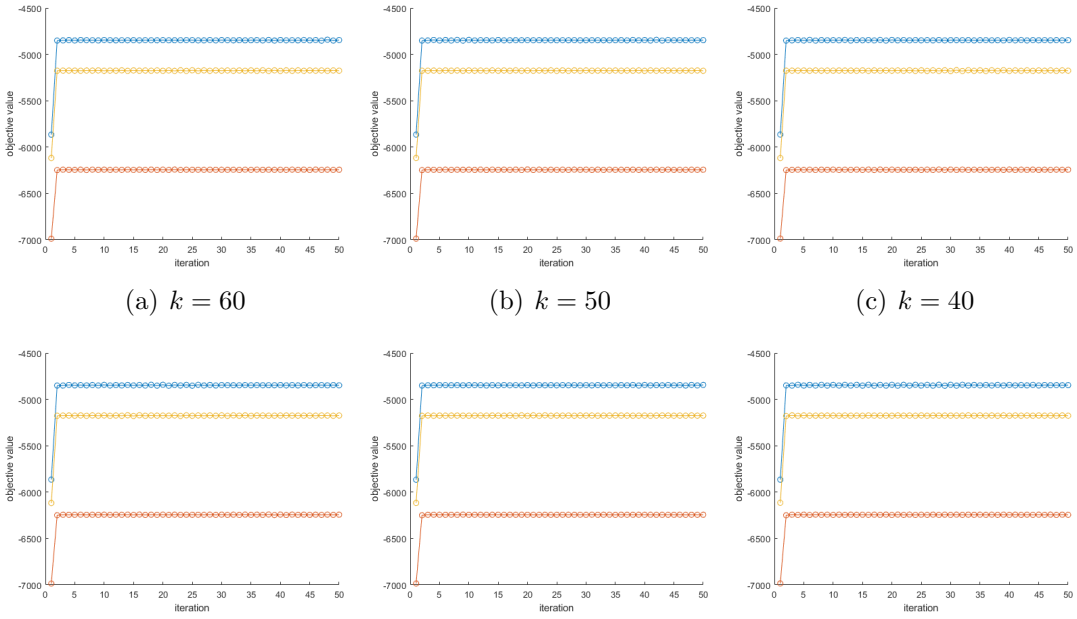
Model \ **Data Selection**		$k = 60$	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
usps	R-OPLS	87.29%	87.29%	87.29%	87.29%	87.29%	87.29%
	SVM	87.84%	87.79%	87.99%	87.99%	87.84%	87.79%

Table 4.7: Accuracy of OVR Multi-Class Classification for **usps** Data Set

protein Data Set

The last multiclass classification experiment with the OVR approach involves the use of a data set named **protein**. Our methodology for this task utilizes the OVR approach, which involves the training of three binary classifiers using the R-OPLS model. Each classifier is trained to classify one class as positive and the remaining two as negative, in their corresponding binary classification problem.

To train the binary classifiers, we use the R-OPLS algorithm to learn the optimal subspace that separates the positive and negative classes for each problem. During training, we experiment with varying the number of subspace dimensions and choose the one that yields the highest classification accuracy on the validation set. Once trained, we evaluate each binary classifier's performance on the test set, where the data is projected into the subspace found during training, and each classifier predicts a class label for each observation. The overall classification accuracy is then calculated as the percentage of correctly classified observations.



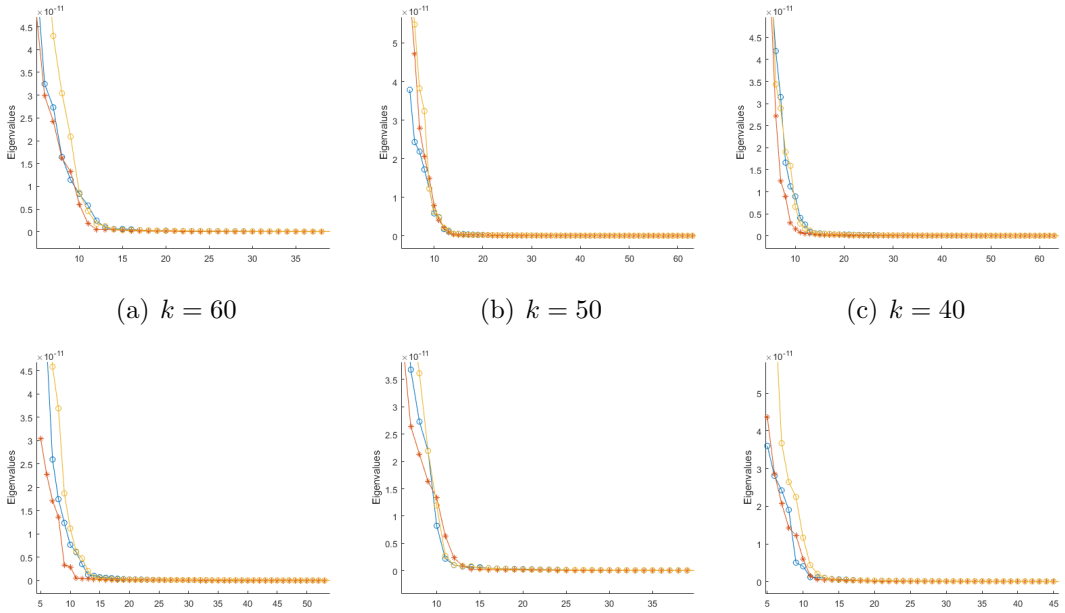
(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

(d) $k = 30$ (e) $k = 20$ (f) $k = 10$

Figure 4.19: OVR Method Iteration plots on **protein** Data

We start by examining the objective function curves of each group of binary classifiers to ensure convergence of the function values to the R-OPLS model. Figure 4.19 displays these curves, which we carefully evaluate to ensure convergence.

After confirming convergence, we use the R-OPLS model for dimensionality reduction. To guarantee optimal performance, we thoroughly analyze the eigenvalue variation curve and find that the eigenvalue curve becomes flatter after $k = 14$. This information is represented in Figure 4.20. The choice of spatial dimension significantly affects classification model performance. For instance, when using the $softmax$ function, the eigenvalue curve converges at a much higher value of $k = 70$ than the optimal value determined using R-OPLS with the OVR method. This result indicates that R-OPLS with OVR method can lead to lower effective dimensionality reduction space and faster convergence of eigenvalue curves. For consistency pur-



(a) $k = 60$ (b) $k = 50$ (c) $k = 40$

(d) $k = 30$ (e) $k = 20$ (f) $k = 10$

Figure 4.20: eigenvalue plots on **protein** Data

poses, we set the value range of k from 10 to 60, with an interval of 10. The Table 4.8 shows the experimental results.

Model \ **Data Selection**		$k = 60$	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
protein	R-OPLS	64.88%	64.88%	64.88%	64.88%	64.88%	64.88%
	SVM	64.49%	64.60%	64.57%	64.99%	64.84%	64.79%

Table 4.8: Accuracy of OVR Multi-Class Classification for **protein** Data Set

Overall, the R-OPLS model combined with the OVR method is used for dimensionality reduction for the **protein** dataset's multiclass classification problem. Our approach leads to faster convergence and improved performance compared to traditional methods such as the $softmax$ function. Additionally, table 4.8 demon-

strates the effectiveness of our method in accurately classifying data into multiple categories.

4.2.3.3 One-vs-One (OVO)

The One-vs-One (OVO) classifier is a popular heuristic method for multi-classification problems that uses binary classification algorithms. This classifier works by combining every two classes in a given dataset into a pair, and then treating each pair as a binary classification task. As a result, there are $c(c-1)/2$ binary classification tasks in the OVO classifier, where c is the number of classes in the data set [18].

When making predictions, test instances are fed into these binary classifiers, which then vote and calculate which class has the most votes [32]. For example, suppose we have a data set with three classes C_1, C_2, and C_3, and we need to create three binary classifiers for each pair of classes, i.e., $f_1,\ f_2,\ f_3$. For each test instance, the classification process involves passing the instance through each binary classifier, and the resulting votes are counted to determine the predicted class.

In the algorithm, the binary classifiers are trained on subsets of the training data that only include examples from the two classes being compared. Then, for a given test instance, each binary classifier produces a vote for one of the two classes. The final predicted class is determined by counting the votes for each class and choosing the one with the highest number of votes.

To perform multi-class classification using the OVO method, we partition the dataset into multiple parts based on the number of classes in the dataset. For instance, the **dna** dataset contains three classes, and we assign X_1 to class 1, X_2 to class 2, and X_3 to class 3. Using this approach, we can obtain three classifiers for the data set as there are $\frac{3(3-1)}{2} = 3$ possible pairs of classes.

Algorithm 4.2 One-vs-One Multi-classification

Require: Training set X, consisting of n examples with d features and k possible class labels

Ensure: Predicted class label for a new input vector x'

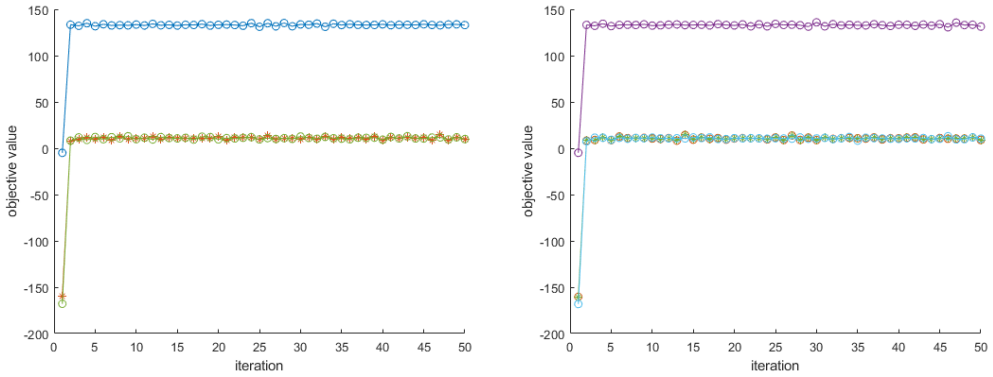
1: **for** $i = 1$ to $k - 1$ **do**
2: **for** $j = i + 1$ to k **do**
3: Construct a new binary training set $X_{i,j}$ by selecting only the examples with class label i or j
4: Train a binary classifier $f_{i,j}(x)$ on $X_{i,j}$
5: **end for**
6: **end for**
7: Count the number of votes for each possible class label by comparing the output of each binary classifier $f_{i,j}(x')$
8: **Return** the class label with the highest number of votes as the predicted label

During the training phase, we use the R-OPLS algorithm to learn all of the binary classifiers that separates the positive (1) and negative classes (-1). After training, we evaluate the performance of each binary classifier on the test set. The test data is projected into the subspace found during training, and each binary classifier is used to predict a class label for each observation. The final classification result is then obtained by combining the results of all the classifiers in the form of voting.

dna Data Set

To determine the optimal parameter combination for the R-OPLS algorithm for the **dna** dataset, we conducted a series of experiments and found that $C = 1$ and $\lambda = 0.5$ provide the best results. We also generated iteration graphs for each

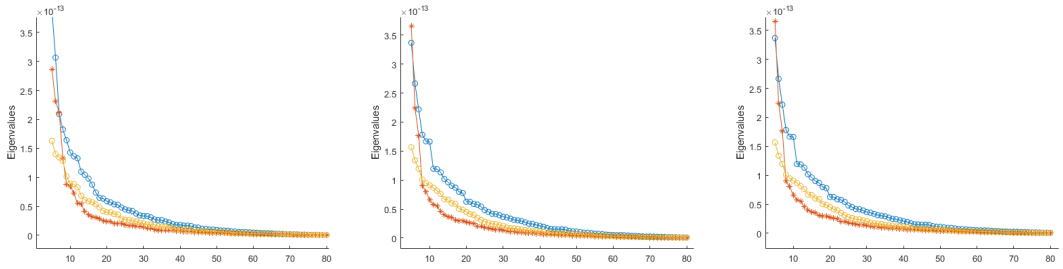
classifier by varying the number of subspace dimensions k during the training phase. From Figure 4.21, all plots are convergent.



(a) $k = 20$ (b) $k = 32$

Figure 4.21: OVO Method Iteration plots on **dna** Data

Additionally, by analyzing the eigenvalue curves in Figure 4.22, we observed that the curves started to converge around $k = 40$. Therefore, for the classification experiment, we chose the value range of k to be from 30 to 50, with an interval of 10, to ensure consistency with the previous experiments. The classification results obtained using the R-OPLS model with the OVO method are presented in Table 4.9.



(a) $k = 20$ (b) $k = 32$ (c) $k = 40$

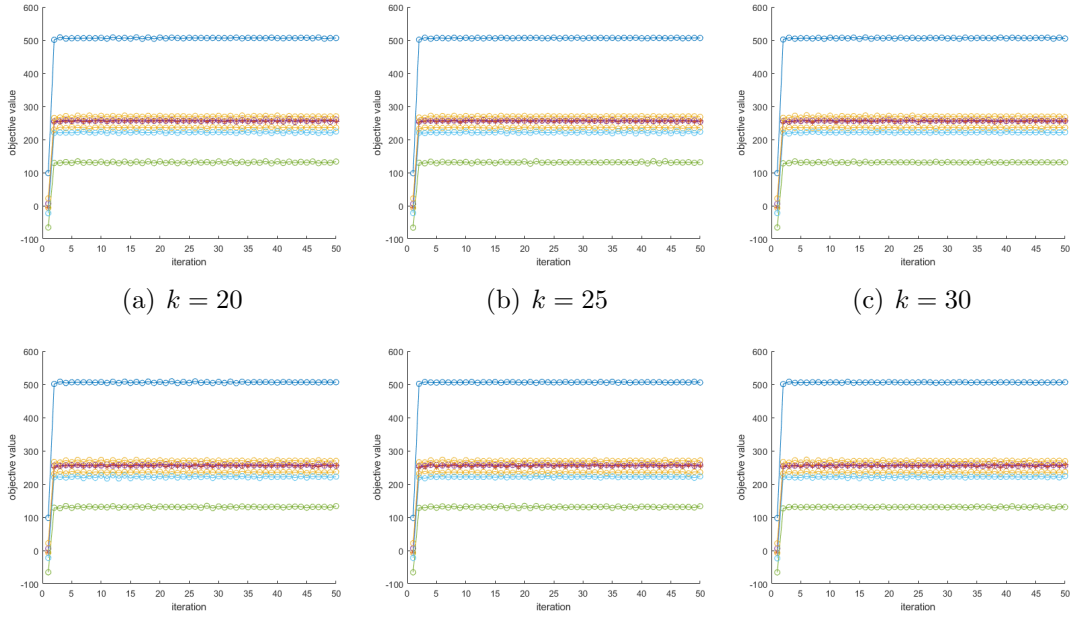
Figure 4.22: eigenvalue plots on **dna** Data

Model \ Data Selection		$k = 60$	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
dna	R-OPLS	86.76%	87.76%	87.76%	87.76%	87.76%	87.76%
	SVM	85.74%	85.08%	85.50%	85.58%	85.58%	84.65%

Table 4.9: Accuracy of OVO Multi-Class Classification for **dna** Data Set

usps Data Set

The usps data set totally has 10 classes, then we also select X_1 with $class_1$, X_2 with $class_2$, ..., X_{10} with $class_{10}$. The number of classifier is $\frac{10(10-1)}{2} = 45$. We also used the input data X_1 and X_2 to train the first classifier, and we reset the labels of X_1 and X_2 as 1 and -1. Follow the same process, the OVO classification approach can obtain 45 classifier results. As we have done previously during testing, each instance is input to 45 classifiers for testing, and then a set of results is obtained in the form of voting.



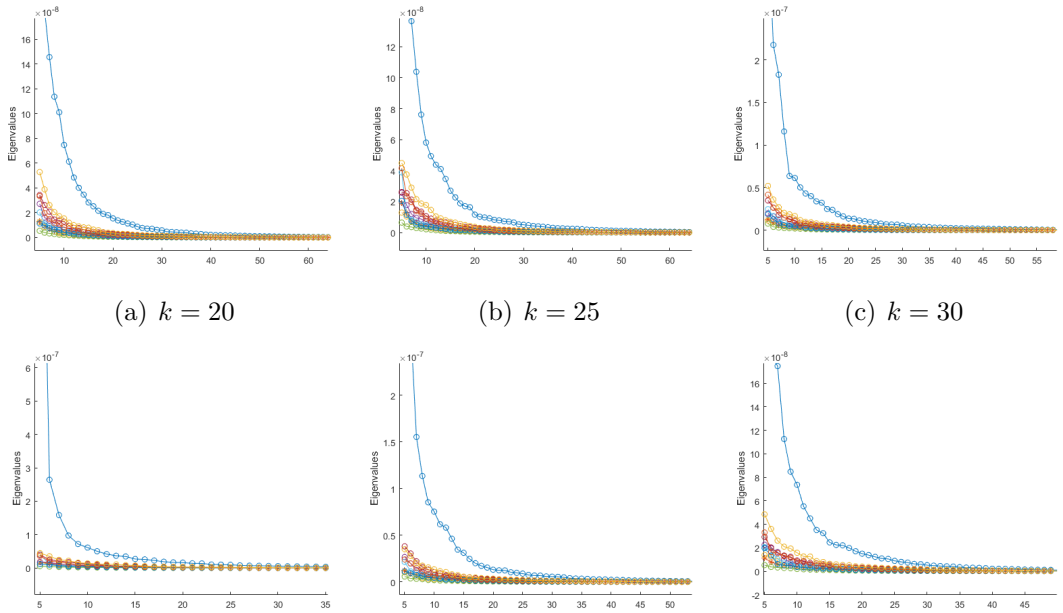
(a) $k = 20$ (b) $k = 25$ (c) $k = 30$

(d) $k = 31$ (e) $k = 35$ (f) $k = 50$

Figure 4.23: OVO Method Iteration plots on **usps** Data

During the **usps** dataset classification experiments using the OVO method, we performed iteration graphs for each classifier by selecting different k values. The graphs revealed that all the curves converge, as demonstrated in Figure 4.23. Since the **usps** data set has ten classes, 45 models need to be created for OVO multi-class classification. In each plot, only the first ten model iteration curves are shown.

To gain insights into the number of effective features per model, we plotted the change graph of the eigenvalue curve in each group of binary classification experiments, as shown in Figure 4.24. The curves began to converge around $k = 31$ in all images, which implies that each model has about 31 effective features. Therefore, the value range of the dimensionality reduction space in the prediction experiment ranges around $k = 31$.



(a) $k = 20$ (b) $k = 25$ (c) $k = 30$

(d) $k = 31$ (e) $k = 35$ (f) $k = 50$

Figure 4.24: eigenvalue plots on **usps** Data

To ensure the consistency of the experiment, we tested the classification accuracy under the same k value selected in the previous experiment, as shown in Table 4.10.

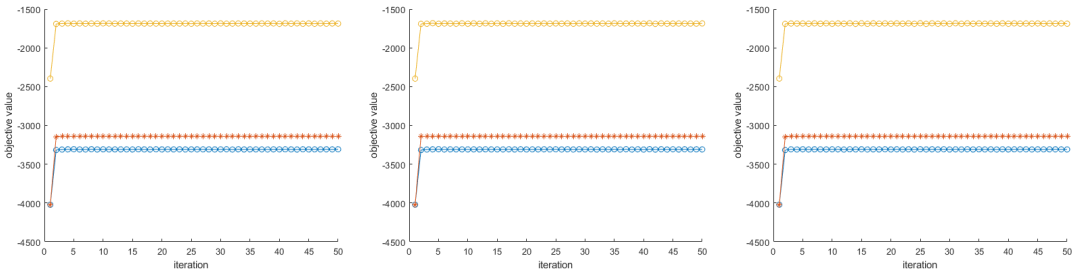
Model \ Data Selection		$k = 60$	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
usps	R-OPLS	88.10%	87.18%	87.47%	88.33%	87.38%	87.79%
	SVM	88.27%	88.34%	87.90%	88.20%	88.75%	88.10%

Table 4.10: Accuracy of OVO Multi-Class Classification for **usps** Data Set

protein Data Set

The **protein** data set have used the same procedure of OVO as the procedure in the **dna** data set. The number of classifier is $\frac{3(3-1)}{2} = 3$. By using the input data X_1 and X_2 to train the first classifier, and we reset the labels of X_1 and X_2 as "1" and "-1". Follow the same process, the OVO classification approach can obtain 3 classifier results. During the testing, each instance is input to three classifiers for testing, and then a set of results is obtained in the form of voting. The classification process is shown as below.

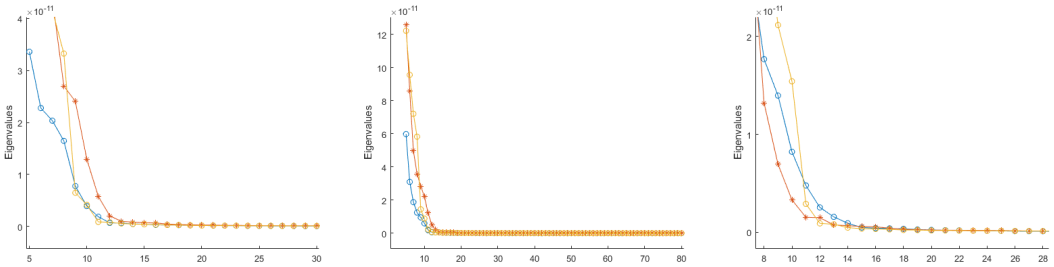
As shown above, the class C_3 has the most results, so the classification prediction result for this testing instant is class C_3.



(a) $k = 20$ (b) $k = 15$ (c) $k = 13$

Figure 4.25: OVO Method Iteration plots on **protein** Data

When select different k, we obtained the eigenvalue plots as shown follows in Figure 4.26. In all images, the curves start to converge around $k = 40$, that is to say, the number of effective features of each model is about 40, so in the prediction experiment, the value range of the dimensionality reduction space is around $k = 40$.



(a) $k = 20$ (b) $k = 15$ (c) $k = 13$

Figure 4.26: eigenvalue plots on **protein** Data

Table 4.11 presents the classification accuracy results with different projected subspace for the **protein** data set. The values are shown for each classifier, as well as the overall accuracy obtained through voting.

Model \ **Data Selection**		$k = 60$	$k = 50$	$k = 40$	$k = 30$	$k = 20$	$k = 10$
protein	R-OPLS	63.66%	63.66%	63.66%	63.66%	63.66%	63.66%
	SVM	63.25%	63.50%	63.28%	63.36%	63.66%	63.36%

Table 4.11: Accuracy of OVO Multi-Class Classification for **protein** Data Set

CHAPTER 5

A Novel Regularized OPLS Model for Multi-view classification Learning

5.1 R-OPLS Model for Multi-view Classification Learning

In practical, a target can be described from many different approaches or different angles, and these different descriptions constitute multiple views. Multi-view data widely exists in the real world and affects all aspects of people's lives. For example, a video can be described as a multi-view data source containing images, audio, and subtitles [46, 68, 70]. A magazine article can be viewed as containing images and text [46, 68]. The hospital's physical examination report also includes multi-view information such as text descriptions, digital indicators, and image descriptions [46, 68, 70]. Data from different views usually contain complementary information, which multi-view learning can exploit to learn more comprehensive representations than single-view learning methods [46, 68, 70]. Due to the increasing use of multi-view data in practical applications, multi-view learning mechanisms have attracted more and more attention from researchers [79]. Since the information representation ability of data largely determines the performance of machine learning methods, multi-view learning becomes a very promising development direction with wide applicability [46]. Multi-view learning [46, 70, 79] is one such learning approach that improves learning performance by exploiting complementary information among multiple views. Many basic algorithms for multi-view learning have been proposed in literature. Assuming that these views are generated from common subspace, the goal of their algorithm is to obtain this common latent subspace shared by all views [79]. The subspace-based learning algorithms [69, 72, 79] have achieved significant

outcomes in many fields, such as speech recognition [46, 79] and bio-informatics [68, 72].

In this thesis, we investigate a novel multi-view learning algorithm formulation, multi-view R-OPLS, that can simultaneously take into account both least-squares-based subspace-based multi-view learning and SVM learning.

The objective of subspace-based learning methods is to obtain a common latent subspace that is shared by multiple views, under the assumption that the input views are generated from this common subspace [70]. Among the subspace-based learning models that already exist, the most typical classic models are canonical correlation analysis (CCA) (Hotelling, 1936) [75], partial least squares (PLS) (H.Wold, 1983) [25], Linear Discriminant Analysis (LDA) (Belhumeur, 1996) [87], and principal component analysis (PCA) (Karl Pearson, 1901) [31]. PCA is a well-known method for reducing the dimensionality of data by projecting it onto a lower-dimensional space while attempting to retain as much of the original data variance as possible [68, 74]. The first principal component, which captures the maximum variance of a specific scalar projection of the data, is placed on the first coordinate. The second largest variance is placed on the second coordinate, and so on [74]. The objective function of PCA is to minimize the sum of squares of distances between the original data points and their projection on the new coordinate system under the constraint that the new coordinates are orthogonal [31, 68]. This can be expressed as finding the eigenvectors and eigenvalues of the covariance matrix of the data, where the eigenvectors correspond to the principal components and the eigenvalues represent the amount of variance explained by each principal component [68]. This technique has been widely used for data analysis and visualization, as well as for feature extraction in machine learning applications, including finance, biology, image processing, and natural language processing [31].

CCA was initially developed to learn two linear projection matrices that maximize the correlation between two views in a shared space [30, 79]. Specifically, CCA has been the popular method as evidenced by its widespread use in vision [41, 68], speech recognition [46, 79], media retrieval [9, 69], and language retrieval [41, 68, 75]. In recent studies, CCA has been extended beyond two views [37, 56, 79] and nonlinear projections by fusing it with other learning modalities, such as supervised learning [68] and kernel CCA [41]. Although CCA and KCCA demonstrate their good effectiveness for modeling the relationship between two or more sets of variables, they still have certain limitations in dealing with associations between multi-view data. Inspired by deep neural networks [46, 66, 77], deep CCA [5] are proposed to address this limitation. And in many studies, based on the basic theory of CCA and the progress of deep neural networks, many researchers have proposed a large number of subspace-based multi-view learning algorithms. Proposals have been made to reformulate the least squares algorithm of CCA for both supervised multi-label classification and unsupervised learning of more than two views, and has been shown to produce effective models and efficient learning algorithms [76, 79]. The approach presented in [69], while effective, is limited to single-view classification as it treats data points as one view and class labels as another [79]. Apart from the CCA series, alternative forms of least squares, including coupled spectral regression [43, 79] and partial least squares (PLS) [45, 79, 83], have been explored for both views.

In contrast to CCA and multi-view CCA, Multi-view Linear Discriminant Analysis (MLDA) [72] employs the within-class scatter matrix instead of the covariance matrix. MLDA is a method used to analyze data from multiple sources or views by reducing the dimensionality of the data [35]. The main objective of multi-view LDA is to find a lower-dimensional subspace that maximizes the separation of classes in each view while also aligning the subspaces across views [15]. This involves identi-

fying the relevant features for discriminating between classes, while considering the correlations and differences between the different views of the data. Compared to traditional LDA, which only handles data with a single view, multi-view LDA considers the covariance matrices of each view separately and then computes a joint covariance matrix that captures the relationships between the views [78]. The joint covariance matrix is then used to determine the projection that maps the data to the lower-dimensional subspace. Multi-view LDA has numerous applications in various domains, such as natural language processing, computer vision, and bioinformatics, where data is often collected from multiple sources or modalities [35, 78].

While the least-squares algorithm has been extensively studied for supervised learning on single-view data, its potential for subspace multi-view learning has been largely overlooked. Like single-view learning, the least squares approach can be also used to express linear discriminant analysis (LDA) for binary classification [9] and multi-class classification [87]. Since CCA has been demonstrated to be equivalent to LDA for multi-class classification [31, 79], the least squares formulation can be shared between CCA and LDA for supervised classification [31, 79, 87]. While CCA and LDA have individual least squares models for two views, obtaining the least squares models for multi-view data is not a simple task. Multi-view discriminant analysis (MvDA) is an extension of regular LDA that supports multi-class classification involving more than two views [5]. Rather than using the least squares approach, multi-view classification techniques are typically formulated as trace ratio problems. However, to simplify numerical treatment, the relaxed ratio trace problem is ultimately solved, resulting in solutions that may not be optimal for the original models [78].

R-OPLS can not only be used for binary classification and multi-class classification under single view, but also can be extended to classification prediction under

multi-view. Next, we describe an experimental extension of R-OPLS under multiple views to reveal the strengths of the proposed model [79].

Given m view labeled data set $\{(x_i^{(1)}, \ldots, x_i^{(m)}, y_i)\},\ i = 1, \ldots, n$, where the ith inputs $x_i^{(s)}$ of all views have two class labels $y_i \in \{-1, +1\}$, R-OPLS propose to seek the projection matrix $P_s \in \mathbb{R}^{d_s \times k}$ to transfer the input data $x_i^{(s)}$ in each view from $\mathbb{R}^{d_s}$ to the common space $\mathbb{R}^k$, where $s = 1, \ldots, m$. Note that the proposed binary classifier will be generalized for multi-class classification via OVR and OVO, which will be explored in our experiment section. Define that the input data points of s view is $X_s = [x_1^{(s)}, \ldots, x_n^{(s)}] \in \mathbb{R}^{d_s \times n}$. Assume that all view classifiers share the same coefficient matrix W [79]. Multi-view R-OPLS can be formulated as minimizing the sum of R-OPLS objectives for all m views, given by

$$\begin{aligned} \min_{P_s, W, v_s} \quad & \sum_{s=1}^{m} \|\tilde{Y} - W^T P_s^T \tilde{X}_s\|_F^2 + \lambda \left[\frac{1}{2}\|v\|_2^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i v^T \hat{x}_i) \right] \\ \text{s.t.} \quad & P_s^T P_s = I_k, \forall k, \\ & \hat{x}_i = [P_1^T \tilde{x}_i^{(1)}; \ldots; P_m^T \tilde{x}_i^{(m)}], \forall i, \end{aligned} \tag{5.1}$$

where $\lambda > 0$ is the regularization parameter. And the label information matrix Y is defined by one-hot representation as shown before, and $\tilde{Y} \in \mathbb{R}^{c \times n}$ and $\tilde{X}_s \in \mathbb{R}^{d_s \times n}$ are matrices transformed from label information matrix Y and input data matrix X_s. For example, the most natural choice is $\tilde{X}_s = \hat{X}_s := X_s H_n, \forall s$, and $\tilde{Y} = \hat{Y} := Y H_n$.

For convenience of mathematics, we express $d = \sum_{s=1}^{m} d_s$ as the total number of features for all m view input data set[79], then we obtain the concatenations of $\{P_s\}$, $\{\tilde{X}_s\}$,and $\{X_s\}$ as shown as follows:

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_v \end{bmatrix} \in \mathbb{R}^{d\times k}, \tilde{X} = \begin{bmatrix} \tilde{X}_1 \\ \tilde{X}_2 \\ \vdots \\ \tilde{X}_m \end{bmatrix} \in \mathbb{R}^{d\times n}, X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} \in \mathbb{R}^{d\times n}. \tag{5.2}$$

Define

$$\tilde{C} = \bar{X}\bar{X}^T = \begin{bmatrix} \tilde{C}_{11} & \tilde{C}_{12} & \dots & \tilde{C}_{1m} \\ \tilde{C}_{21} & \tilde{C}_{22} & \dots & \tilde{C}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{C}_{m1} & \tilde{C}_{m2} & \dots & \tilde{C}_{vm} \end{bmatrix} \in \mathbb{R}^{d\times d}, \tag{5.3}$$

and its diagonal block part is

$$\tilde{C}_{diag} = \begin{bmatrix} \bar{C}_{11} & & & \\ & \bar{C}_{22} & & \\ & & \ddots & \\ & & & \bar{C}_{mm} \end{bmatrix} \in \mathbb{R}^{d\times d}, \tag{5.4}$$

where $\tilde{C}_{s,t} = \tilde{X}_s \tilde{X}_t^{\ T}, \forall s, t = 1, \dots, m$.

From the OPLS mathematical parts, we obtained the first order optimality condition of formula (5.1) with respect to W as shown follows:

$$\sum_{s=1}^{m} -2P_s^T \tilde{X}_s (\tilde{Y} - W^T P_s^T \tilde{X}_s)^T = 0, \tag{5.5}$$

and then the optimal solution of W is [79]

$$W^* = (\sum_{s=1}^{m} P_s^T \tilde{C}_{ss} P_s)^{-1} \sum_{s=1}^{m} P_s^T \tilde{X}_s \tilde{Y}^T = (P^T \tilde{C}_{diag} P)^{-1} P^T \tilde{X} \tilde{Y}^T. \tag{5.6}$$

Then, following the same procedure we did in section 3.1, by substituting the optimal W back into formula (5.1), and trading off square losses and hinge losses, we can reformulate the multi-view problem as

$$\max_{\boldsymbol{\alpha}} \min_{P} \ -\mathrm{tr}((P^T(\tilde{C}_{diag}+\epsilon I_d)P)^{-1}P^T\tilde{X}\tilde{Y}^T\tilde{Y}\tilde{X}^TP)-\frac{\lambda}{2}\mathrm{tr}(\Omega^{-1}P^T\tilde{X}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\tilde{X}^TP)+\lambda\mathbf{1}_n^T\boldsymbol{\alpha}$$

$$\text{s.t.} \ \ P^TP = I_k, \tag{5.7}$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n.$$

To streamline and enhance the optimization process, we also apply the identical methodologies detailed in Subsection 2.2. The primary objective behind the design of the flexibility matrix Ω was to ensure its numerical treatment is simple, while also ensuring that the properties of Multi-view R-OPLS are preserved. For example, when $\Omega = P^T(\tilde{C}_{diag} + \epsilon I_d)P$, the problem (5.1) of Multi-view R-OPLS is equivalent to

$$\max_{\boldsymbol{\alpha}} \min_{P} \ \ -tr(P^T\tilde{X}[\tilde{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T]\tilde{X}^TP) + \lambda\mathbf{1}_n^T\boldsymbol{\alpha} \tag{5.8}$$

$$\text{s.t.} \ \ P^T\tilde{C}_{diag}P = I_k,$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n.$$

To address the optimization problem (5.8), we once again decompose it into two sub-problems concerning the unknowns P and $\boldsymbol{\alpha}$, mirroring the approach taken in Section 3.1.

5.2 Algorithms

The alternating method requires an initial value to start. To obtain the initial value $\boldsymbol{\alpha}$, we solve the following problem

$$\boldsymbol{\alpha_0} = \arg\min_{\boldsymbol{\alpha}} \ \ \frac{1}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\tilde{X}^T\tilde{X}(\boldsymbol{\alpha}\odot\boldsymbol{y}) - \mathbf{1}_n^T\boldsymbol{\alpha} : \text{s.t.} \ \ \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n. \tag{5.9}$$

Problem (5.9) is a quadratic programming problem with box constraint. It can be solved by *quadprog* solver in MATLAB.

Given $\boldsymbol{\alpha}$, we obtain the optimal P by solving

$$\max_{P} \ tr(P^T Q P) : \text{s.t. } P^T(\tilde{C}_{diag} + \epsilon I_d)P = I_k, \tag{5.10}$$

where $Q = \tilde{X}[\tilde{Y}^T\tilde{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T]\tilde{X}^T \in \mathbb{R}^{d\times d}$.

The problem (5.10) is a generalized eigenvalue problem [26] on matrix Q with $(\tilde{C}_{diag} + \epsilon I_d)$. The optimal $\hat{P}$ is the eigenvectors of Q with $(\tilde{C}_{diag} + \epsilon I_d)$ corresponding to top k eigenvalues [26]. The Lagrange multiplier is diagonal in the eigenvalue problem.

In the alternating iterative experiments of multi-view R-OPLS, we utilized the same approach as described in Section 3.1. We also employed two distinct methods to solve $\boldsymbol{\alpha}$, namely the quadratic programming solver (*quadprog*) and the projected gradient descent method (PGD). The results of these experiments were then compared to evaluate the effectiveness of each method.

Given P, we obtain the optimal $\boldsymbol{\alpha}$ by solving

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})^T \tilde{X}^T P P^T \tilde{X}(\boldsymbol{\alpha} \odot \boldsymbol{y}) - \mathbf{1}_n^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{5.11}$$

Then, denoting that $\tilde{B} = (\tilde{X}^T P P^T \tilde{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T) \in \mathbb{R}^{n\times n}$ is positive definite, we have the following optimization problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T \tilde{B}\boldsymbol{\alpha} - \mathbf{1}_n^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n. \end{aligned} \tag{5.12}$$

When *quadprogs* is used with multi-view R-OPLS, the process of alternately iteratively solving $\boldsymbol{\alpha}$ and P is shown in Algorithm 5.1. At each iteration, *quadprog* is used to solve for the updated values of $\boldsymbol{\alpha}$, while P is updated using the standard generalized eigenvalue problem solver.

Algorithm 5.1 Multi-view R-OPLS Iteration with Quadratic Programming Solver

1: **Initialization:** solving $\boldsymbol{\alpha}^{(0)}$ by (5.9)

2: **for** i=1:50 **do**

3: Update $P^{(i)}$ based on (5.10)

4: Update $\boldsymbol{\alpha}^{(i)}$ based on (5.12)

5: **end for**

6: **Output**: $\boldsymbol{\alpha}$ and P

In addition to using the quadratic programming solver, we can also use the projected gradient descent method (PGD) [8] to obtain the optimal solution for $\boldsymbol{\alpha}$. This method involves reformulating the problem as follows:

$$\min_{\boldsymbol{\alpha}\in\mathcal{A}} \; g(\boldsymbol{\alpha}) := \max_{P} \; \mathrm{tr}(P^T\hat{X}[\hat{Y}^T\hat{Y} + \frac{\lambda}{2}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T]\hat{X}^TP) - \lambda\mathbf{1}_n^T\boldsymbol{\alpha}, \tag{5.13}$$

where $\mathcal{A}$ represents the domain of $\boldsymbol{\alpha}$, such as $\mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1}_n$.

In our model, the gradient descent expression is

$$\boldsymbol{\alpha}^{(t+\frac{1}{2})} \leftarrow \boldsymbol{\alpha}^{(t)} - \lambda[((\hat{X}^TPP^T\hat{X}) \odot (\boldsymbol{y}\boldsymbol{y}^T))\boldsymbol{\alpha} - \mathbf{1}_n]. \tag{5.14}$$

And the projection area is

- Projection

$$\alpha^{(t+1)} = \begin{cases} \alpha^{(t+\frac{1}{2})}, & \alpha^{(t+\frac{1}{2})} \in [0,\ C]; \\ 0, & \alpha^{(t+\frac{1}{2})} < 0; \\ C, & \alpha^{(t+\frac{1}{2})} > C. \end{cases} \tag{5.15}$$

Algorithm 5.2 Multi-view R-OPLS Iteration with Projected Gradient Descent

Initialization: Solve (5.9) for $\boldsymbol{\alpha_0}$
for i=1 to 50 **do**
    Update $P^{(i)}$ based on (5.10)
    Update $\alpha^{(i+\frac{1}{2})} \leftarrow \alpha^{(i)} - \lambda \nabla_\alpha g(\alpha^{(t)})$
    if $\alpha^{(i+\frac{1}{2})} \in [0,\ C]$ **then**
        $\alpha^{(i+1)} = \alpha^{(i+\frac{1}{2})}$
    else
        if $\alpha^{(i+\frac{1}{2})} < 0$ **then**
            $\alpha^{(i+1)} = 0$
        else
            $\alpha^{(i+1)} = C$
        end if
    end if
end for
Output: $\boldsymbol{\alpha}$ and P

Algorithm 5.2 describes the iterative process of solving for α and P alternately when multi-view R-OPLS is combined with the projected gradient descent method.

5.3 Generalized multi-view R-OPLS framework

The Multi-view R-OPLS (5.1) algorithm can be represented as a fusion of least squares and support vector machines (SVM). Consequently, it is possible to employ regularization techniques to fine-tune model parameters and incorporate prior knowledge into the algorithm. This approach can help to shape the similarity be-

tween projection points and ultimately reduce or eliminate the heterogeneity gap that may exist between different views. By the effective use of popular regularization techniques in computational sciences to regulate solutions of inverse problems, along with the success of combining these techniques with orthonormalized partial least squares [79, 83], we present a generalized Multi-view R-OPLS framework.

$$\begin{aligned}\min_{P_s,W,v_s} \quad & \sum_{s=1}^{m}\|\tilde{Y}-W^TP_s^T\tilde{X}_s\|_F^2-\frac{\lambda_1}{2}\mathrm{tr}(\Omega^{-1}P^T\tilde{X}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\tilde{X}^TP)+\lambda_1\mathbf{1}_n^T\boldsymbol{\alpha} \\ & +\lambda_2\mathrm{tr}(W^TP_s^TAP_sW)+\lambda_3\mathrm{tr}(\Omega^{-1}P_s^TBP_s) \\ \text{s.t.} \quad & P_s^TP_s=I_k, \\ & \mathbf{0}\leq\boldsymbol{\alpha}\leq C\mathbf{1}_n,\end{aligned} \tag{5.16}$$

where $A\in\mathbb{R}^{d\times d}$ is a symmetric square matrix and $B\in\mathbb{R}^{d\times d}$ is a symmetric definite matrix [68]. That is, all of the eigenvalue of B are not equal to 0. The primary objective behind the design of the flexibility matrix Ω was to ensure its numerical treatment is simple, while also ensuring that the properties of Multi-view R-OPLS are preserved [79]. For example, when $\Omega=P^T(\tilde{C}_{diag}+A)P$, the problem (5.16) has the analytic solution for W as follows

$$W=(P^T(\tilde{C}_{diag}+\lambda_2A)P)^{-1}P^T\tilde{X}\tilde{Y}^T. \tag{5.17}$$

Then, putting the analytic solution for W back into (5.16), we obtained the objective function as

$$\begin{aligned}&\|\tilde{Y}\|_F^2-\mathrm{tr}((P^T(\tilde{C}_{diag}+\lambda_2A)P)^{-1}P^T\tilde{X}\tilde{Y}^T\tilde{Y}\tilde{X}^TP)-\frac{\lambda_1}{2}\mathrm{tr}(\Omega^{-1}P^T\tilde{X}(\boldsymbol{\alpha}\odot\boldsymbol{y})(\boldsymbol{\alpha}\odot\boldsymbol{y})^T\tilde{X}^TP) \\ &+\lambda_1\mathbf{1}_n^T\boldsymbol{\alpha}+\lambda_3\mathrm{tr}(\Omega^{-1}P^TBP).\end{aligned} \tag{5.18}$$

By discarding the constant term and taking into account the property of trace, we derive the framework of generalized multi-view R-OPLS as

$$\max_{\boldsymbol{\alpha}} \min_{P} \quad -\mathrm{tr}(P^T[\tilde{X}[\tilde{Y}^T\hat{Y} + \frac{\lambda_1}{2}(\boldsymbol{\alpha} \odot \boldsymbol{y})(\boldsymbol{\alpha} \odot \boldsymbol{y})^T]\tilde{X}^T - \lambda_3 B]P) + \lambda_1 \mathbf{1}_n^T \boldsymbol{\alpha} \quad (5.19)$$

$$\text{s.t.} \quad P^T(\tilde{C}_{diag} + \lambda_2 A)P = I_k,$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}_n.$$

It is important to note that providing valid values of Ω is necessary to ensure that the generalized multi-view R-OPLS equation (5.16) can be equivalent to the generalized eigenvalue problem. In such cases, finding a numerical solution for formula (5.16) can be a subject of further investigation. However, in the remaining sections of the thesis, all models instantiated in (5.16), including pre-existing models, assume that Ω is given.

The addition in (5.16) does not compromise any of the properties listed in Multi-view R-OPLS; rather, it improves the model's capacity to integrate priors for optimal learning. Furthermore, the regularization form of Multi-view R-OPLS can take other forms besides (5.16). Other regularization techniques such as orthographic projection for distance preservation which is mentioned in [89] and $l_{2,1}$ norm over P for feature selection with sparse subspace learning which is mentioned in [86] are also viable options. However, these approaches result in different solution structures and require distinct models compared to the numerical techniques used to solve GEP [79]. Therefore, we will not explore them in detail in this thesis.

5.4 Examples of generalized multi-view R-OPLS

5.4.1 Multi-view CCA

It should be noted that multi-view CCA (MCCA) [37, 56, 76] is a subspace learning method that operates in an unsupervised manner. Since the labels of the data are unknown, it is convenient to assume that each instance belongs to a distinct

class, resulting in $c = n$. To transform unlabeled data into labeled data, we can assign each instance a unique class label, which is represented by $Y = I_n$. If we assign $\lambda_1 = \lambda_2 = \lambda_3 = 0$ in generalized multi-view R-OPLS (5.16), while also utilizing $\tilde{X} = \hat{X} = XH_n, \tilde{Y} = I_n$, the resulting model will be equivalent to MCCA.

$$\begin{aligned} \min_{P_s,W} \quad & \sum_{s=1}^{m} \|\tilde{Y} - W^T P_s^T \tilde{X}_s\|_F^2 := \max_{P} \sum_{s=1}^{m} \sum_{t=1}^{m} \mathrm{tr}(P^T \hat{C}_{st} P) \\ \text{s.t.} \quad & \sum_{s=1}^{m} P_s^T (\hat{C}_{ss} + \beta I_{d_s}) P_s = I_k, \end{aligned} \tag{5.20}$$

where $\hat{C}_{st} = X_s H_n X_t^T, \forall s, t = 1, \ldots, m$.

5.4.2 LDA

The objective of LDA [72] is to discover a shared subspace that can maximize the correlation between two views and optimize the discriminative power of each view at the same time. As same as the previous multi-view CCA section, multi-view LDA is also a special case of generalized multi-view R-OPLS with $\tilde{X} = \hat{X} = XH_n$, and $\tilde{Y} = \Sigma^{-1/2} Y$,while also using $\lambda_1 = \lambda_3 = 0$, and $A = \Gamma$.

$$\Gamma = \begin{bmatrix} \gamma_1 I_{d_1} & & & \\ & \gamma_2 I_{d_2} & & \\ & & \ddots & \\ & & & \gamma_m I_{d_m} \end{bmatrix} \in \mathbb{R}^{d \times d},$$

where $\gamma_s \geq 0$ is the weight for view s and the block diagonal matrix [79].

Then, we can obtain the problem

$$\begin{aligned} \min_{P,W} \quad & \sum_{s=1}^{m} \|\tilde{Y} - W^T P^T \tilde{X}\|_F^2 := \max_{P} \mathrm{tr}(P^T S_b P) \\ \text{s.t.} \quad & P^T (XH_n X^T + \beta I_d) P = I_k, \end{aligned} \tag{5.21}$$

where $S_b = X(\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T)X^T$ is the between-class scatter matrix [72].

For multi-class classification, we defined the counting matrix Σ of class labels

$$\Sigma = \begin{bmatrix} n_1 & & & \\ & n_2 & & \\ & & \ddots & \\ & & & n_c \end{bmatrix} = YY^T \in \mathbb{R}^{c\times c}, \tag{5.22}$$

where $n_r = \sum_{i=1}^{n} Y_{r,i}$ denotes the total number of data points in class r [79].

Then, we can obtain the similarity symmetric matrix $\Psi = Y^T\Sigma^{-1}Y$, and

$$\Psi\mathbf{1}_n = Y^T\Sigma^{-1}\begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_c \end{bmatrix} = Y^T\mathbf{1}_c = \mathbf{1}_n. \tag{5.23}$$

$\hat{\Psi}$ is the centered matrix of Ψ, and is given by

$$\hat{\Psi} = H_n\Psi H_n = (I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T)\Psi(I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T) \tag{5.24}$$

$$= \Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\Psi - \Psi\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T + \frac{1}{n^2}\mathbf{1}_n\mathbf{1}_n^T\mathbf{1}_n\mathbf{1}_n^T \tag{5.25}$$

$$= \Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T. \tag{5.26}$$

We will illustrate that supervised classification via multi-view LDA is a particular form of generalized multi-view R-OPLS that utilizes $\tilde{X} = \hat{X} = XH_n$ and $\tilde{Y} = \Sigma^{-1/2}Y$.

$$\tilde{X}\tilde{Y}^T\tilde{Y}\tilde{X}^T = XH_nY^T[\Sigma^{-1/2}]^T[\Sigma^{-1/2}]YH_nX^T$$

$$= XH_nY^T\Sigma^{-1}YH_nX^T$$

$$
\begin{aligned}
&= XH_n\Psi H_n X^T \\
&= X(\Psi - \frac{1}{n}\mathbf{1}\mathbf{1}^T)X^T \\
&:= S_b. \qquad (5.27)
\end{aligned}
$$

5.4.3 Multi-view PCA

The goal of PCA [31, 68] is to minimize the total squared distance between the original data points and their corresponding projections onto a new coordinate system, while ensuring that the new coordinate system is orthogonal [60, 74]. Given λ_2 and symmetric matrix A so that it satisfies $\tilde{C}_{diag} + \lambda_2 A = I_d$, then combined with $\lambda_1 = 0$ and $\lambda_3 = 0$, the generalized multi-view R-OPLS formula (5.16) is equivalent to PCA.

$$
\begin{aligned}
\min_{P_s, W} \quad & \sum_{s=1}^{m} \|\tilde{Y} - W^T P_s^T \tilde{X}_s\|_F^2 + \lambda_2 \mathrm{tr}(W^T P_s^T A P_s W) \qquad (5.28) \\
\text{s.t.} \quad & P_s^T P_s = I_k.
\end{aligned}
$$

Then, It is equivalent to

$$
\max_{P} \quad \mathrm{tr}(P^T \tilde{X} \tilde{Y}^T \tilde{Y} \tilde{X}^T P) : \text{s.t. } P^T P = I_k, \qquad (5.29)
$$

where $\tilde{X} = \hat{X} = XH_n$, and $\tilde{Y} = I_n$.

5.4.4 MLDA

MLDA [72] aims to learn a shared space that maximizes the correlation between two views and the discriminative power of each view at the same time. One way to express MLDA is as a special case of generalized multi-view R-OPLS, which is achieved by setting $\tilde{X} = \hat{X} = XH_n, \tilde{Y} = I_n$, and choosing appropriate values for the matrices A and B in Equation (5.16).

Let $\lambda_1 = \lambda_2 = 0$ and $B = X_s R X_s^T$, where $R = H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T), \forall s = 1, \ldots, m$, the objective function of (5.16) can be rewritten as

$$\min_{P_s, W} \sum_{s=1}^{m} \|\tilde{Y} - W^T P_s^T \tilde{X}_s\|_F^2 + \lambda_3 \mathrm{tr}(\Omega^{-1} P_s^T X_s (H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T)) X_s^T P_s) \quad (5.30)$$

Then, It is equivalent to

$$\max_{P} \ \mathrm{tr}(P^T K P) : \text{s.t. } P^T \hat{C}_{diag} P = I_k, \quad (5.31)$$

where K is a block matrix with the $(st)th$ blocks,

$$S_{st} = \begin{cases} \lambda_3 X_s (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T) X_s^T : \ s = t, \\ \hat{C}_{st} : \qquad\qquad\qquad otherwise. \end{cases}$$

The between-class scatter matrix with scaling λ_3 is $\tilde{C}_{ss} - X_s R_s X_s^T = \hat{C}_{ss} - X_s(H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T))X_s^T = \tilde{X}_s(\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T)$. Therefore, the formula (5.31) is equivalent to MLDA.

5.4.5 GMA

GMA [68] aims to learn a shared subspace across all views, capturing the common information and reducing the dimensionality of the data. The objective of GMA is to minimize the sum of reconstruction errors for each view while preserving the shared structure of the learned subspace [31, 79]. By merging discrimination information and cross-view correlations, GMA directly applies kernel techniques to learn non-linear transformations for more than two views [68, 74]. The resulting common subspace allows the analysis of data across all views, leading to improved performance in downstream tasks such as classification and clustering [3, 68, 74]. The optimization problem can be solved by using the alternating direction method of multiplication (ADMM) algorithm [3, 74]. The learned subspace and coefficients can be used for various downstream tasks such as clustering, classification, and visualization.

A key difference between GMA and MLDA lies in their constraints. The constraint of MLDA is using the total scatter, but the constraint of GMA is using within-class scatter. Therefore, based on the problem (5.30), and setting $A = X_s H_n Y^T \Sigma^{-1} Y H_n X_s^T$, we can obtain the problem

$$\min_{P_s,W} \sum_{s=1}^{m} \|\tilde{Y} - W^T P_s^T \tilde{X}_s\|_F^2 + \lambda_2 \mathrm{tr}(W^T P_s^T A P_s W) + \lambda_3 \mathrm{tr}(\Omega^{-1} P_s^T X_s (H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T)) X_s^T P_s), \tag{5.32}$$

with $\tilde{X} = \hat{X} = XH_n, \tilde{Y} = I_n$, and $\Omega = \sum_{s=1}^{m} \mathrm{tr}(P_s^T X_s (I_n - \Psi) X_s^T P_s)$. Then problem (5.32) is equivalent to

$$\max_{P} \ \mathrm{tr}(P^T K P) : \text{s.t.} \ \sum_{s=1}^{m} \mathrm{tr}(P_s^T X_s (I_n - \Psi) X_s^T P_s) = I_k. \tag{5.33}$$

Therefore, we can summary that

- **MCCA**: $\tilde{X} = \hat{X} = XH_n,\ \tilde{Y} = I_n,\ A = 0,\ B = 0.$
- **LDA**: $\tilde{X} = \hat{X} = XH_n,\ \tilde{Y} = \Sigma^{-1/2} Y,\ A = \Gamma,\ B = 0.$
- **MPCA**: $\tilde{X} = \hat{X} = XH_n,\ \tilde{Y} = I_n,\ \tilde{C}_{diag} + \lambda_2 A = I_d,\ B = 0.$
- **MLDA**: $\tilde{X} = \hat{X} = XH_n,\ \tilde{Y} = I_n,\ A = 0,\ B = X_s(H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T))X_s^T.$
- **GMA**: $\tilde{X} = \hat{X} = XH_n,\ \tilde{Y} = I_n,\ A = X_s H_n Y^T \Sigma^{-1} Y H_n X_s^T,\ B = X_s(H_n - (\Psi - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T))X_s^T.$

We will not list all possible variants due to the numerous combinations involved. For the task at hand, we suggest choosing or devising a suitable regularization term to integrate into the proposed regularized Multi-view R-OPLS framework (5.16).

5.5 Numerical Experiments

5.5.1 Data Information

The Multiple Features (**Mfeat**) data set is used for multi-view feature evaluated by multi-class classification. The **Mfeat** data set is obtained from UCI's machine learning repository, where the introduction of every views can be found there.

In the **Mfeat** data set, 2000 instances are chosen in the experiments, and it has 10 classes which are handwritten numbers $(0-9)$ with 200 pictures in each class. These numbers are expressed in 6 different types of features (heterogeneous features) with different dimensionalities [21, 51, 79, 80]. The 6 views are fou: Fourier coefficients of the character shapes with 76 features, fac: profile correlations with 64 features, kar: Karhunen-Loeve coefficients with 64 features, mor: morphological features with 6 features, zer: Zernike moments with 47 features, and pix: pixel average with 240 features [21, 51, 79, 80].

Data Set	**Views**	**Classes**	**Total Number of Features**	**Training**	**Testing**
Mfeat	6	10	180	1600	400

Table 5.1: Multi-view Data Information

5.5.2 Numerical Experiments

The objective of this experiment is to evaluate the effectiveness of multi-view R-OPLS in comparison to other baseline multi-view learning models in classifying datasets with multiple views. The process involves preprocessing the data and splitting it into training and testing sets for model training and evaluation. The performance metrics used to compare the models include classification accuracy. The comparison models may include other well-known multi-view learning techniques such as MCCA [76], MPCA [31], MLDA [72], and GMA [68]. In all of the methods used in this experiment, the goal is to learn a set of linear projections that can transform the data points of each view into points in a low-dimensional common subspace. The common subspace is then used to perform classification experiments on the data.

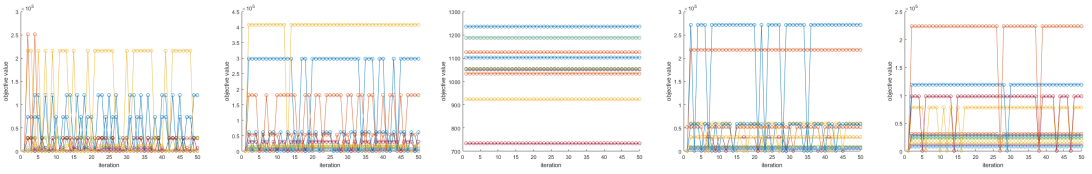
For this experiment, the first step is to partition the data set based on its different views. Taking the Mfeat data set as an example, we represent the entire data set as X and then divide it into six subgroups, X_1 to X_6, and each subgroup represents a specific view. For instance, X_1 corresponds to the data in the fac view, X_2 represents the data in the fou view, X_3 represents the data in the kar view, X_4 represents the data in the mor view, X_5 represents the data in the pix view, and X_6 corresponds to the data in the zer view. Next, the data in each group is further divided into a training set and a testing set. Several different methods are then applied to learn the projection matrix P from the training set and obtain a new representation of the data on a low-dimensional subspace.

The common parameters shared by Multi-view R-OPLS and its variants include the SVM penalty parameter C, the reduced dimension parameter k, and SVM regularization parameter λ_1. Additionally, other methods such as MCCA, MLDA, MPCA, GMA have an extra tuning parameter $\lambda_2 = \lambda_3 = 10^{-3}$ for an additional regularization term. To simplify the experiments, we also set $\gamma_s = \gamma = 10^{-6}$ in the matrix Γ for all experiments. The parameter k is crucial for all subspace learning methods, and we select it by observing the eigenvalue plot. We used a quadratic programming solver and a generalized eigenvalue problem solver during training.

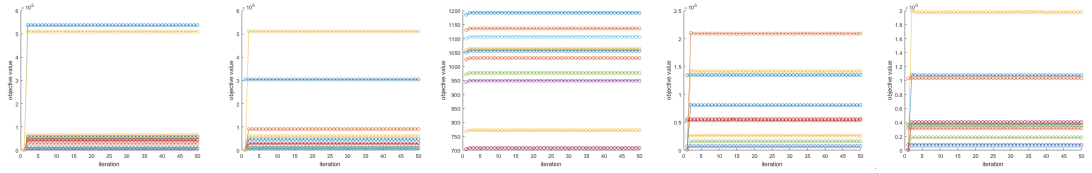
In the study of multi-view R-OPLS, we investigated the performance of two multi-classification methods, namely OVR and OVO, on the **Mfeat** data set which is multi-view and multi-class. To solve for the weight vector $\boldsymbol{\alpha}$, we used both quadprog solver and projected gradient descent in alternate iterative experiments for each method.

The experiments conducted using the multi-view and multi-class **Mfeat** data set involved the use of two different multi-classification methods, namely OVR and OVO. The objective function curve for the OVR method using the *quadprog* solver

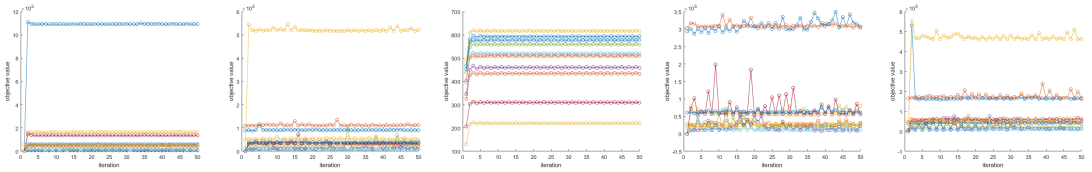
is displayed in Figure 5.1, which shows that the convergence of the objective function is influenced by the values of the parameters C and λ. The results indicate that the variation of the function value is the smallest when $\lambda = 0.5$



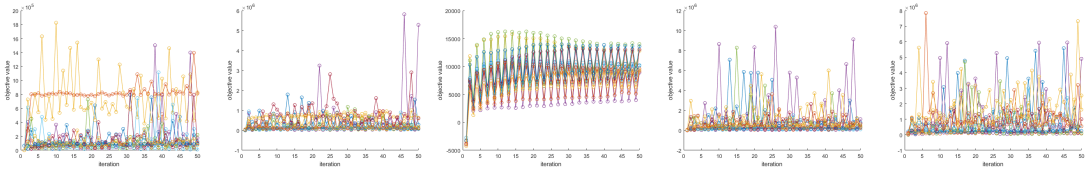
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



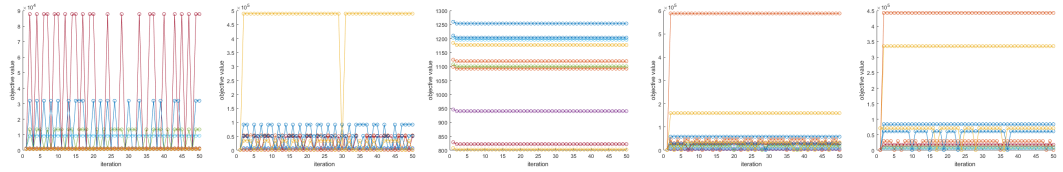
(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



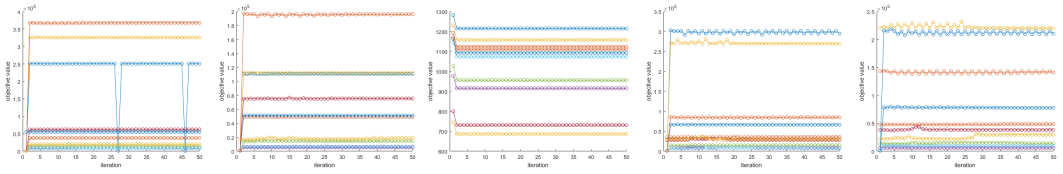
(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 5.1: The plots of OVR with *quadprog* Solver on **mfeat** Data Set

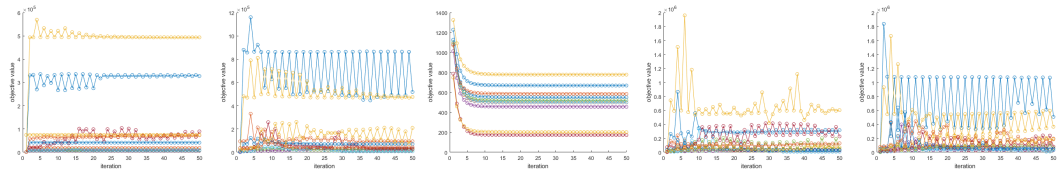
Furthermore, the projected gradient descent method was utilized to solve for $\boldsymbol{\alpha}$, and Figure 5.2 presents the change curve of the objective function value under the OVR method. The results indicate that the objective function values can smoothly decrease and reach the minimum value when $C = 1$ and $\lambda = 0.5$.



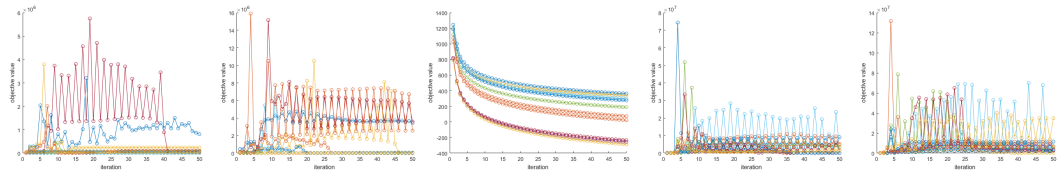
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$

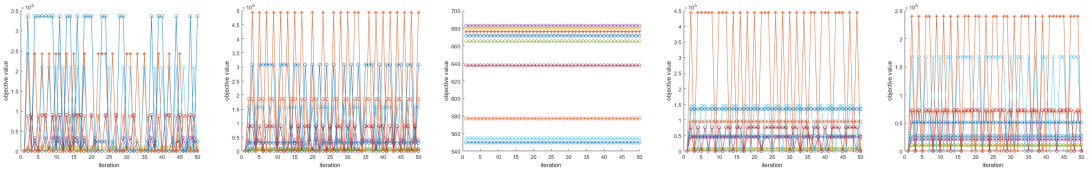


(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

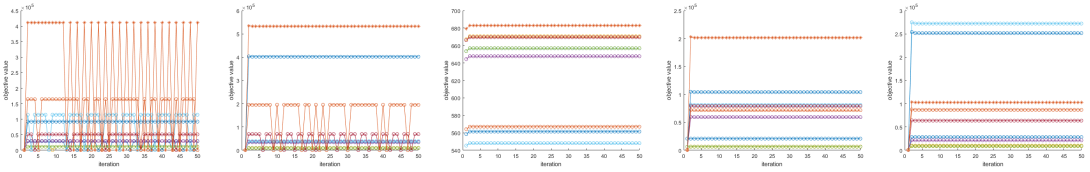
Figure 5.2: The plots of OVR with PGD Method on **mfeat** Data Set

We also conducted experiments using quadprog solver and projected gradient descent to analyze the effectiveness of different methods in the OVO method. The corresponding change curves of the objective function values are shown in Figure 5.3 and Figure 5.4, respectively. Figure 5.3 displays the change curve of the objective function value when using the quadprog solver under the OVO method, while Figure 5.4 presents the change curve of the objective function when using the projected gradient descent method under the OVO method. The results obtained by these two methods are consistent with those obtained by the OVR method.

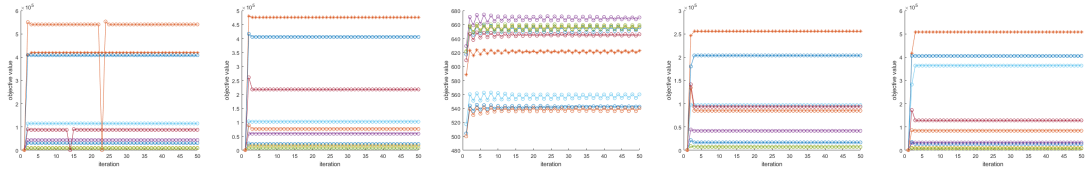
It is important to note that since the mfeat data has 10 classes, the OVO method consists of 45 pairwise binary classifications. For clarity, we chose only 10 pairwise classifications for each parameter combination to plot the curves. Overall, our results demonstrate the effectiveness of both OVR and OVO methods in multi-classification tasks and the impact of different optimization algorithms on the convergence of the objective function.



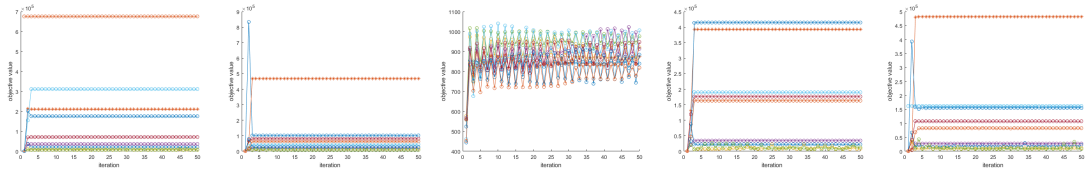
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$

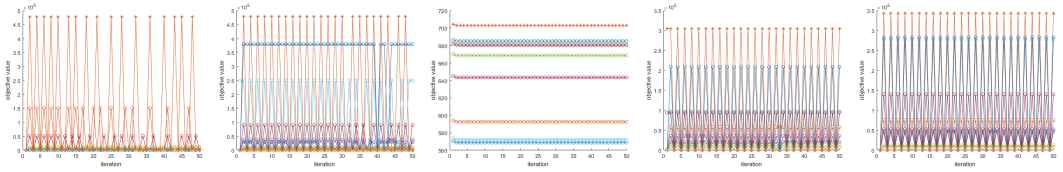


(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

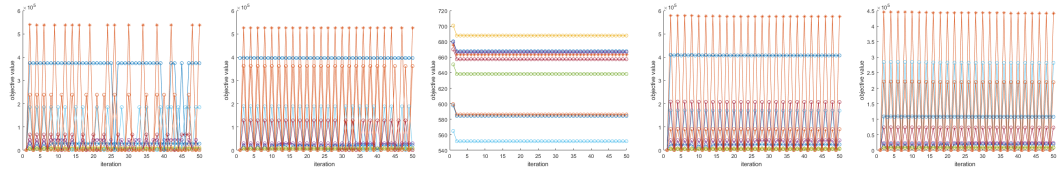
Figure 5.3: The plots of OVO with *quadprog* Solver on **mfeat** Data Set

The experimental results presented in Figures 5.1, 5.2, 5.3, and 5.4 demonstrate the effectiveness of the proposed multi-view R-OPLS method in solving multi-

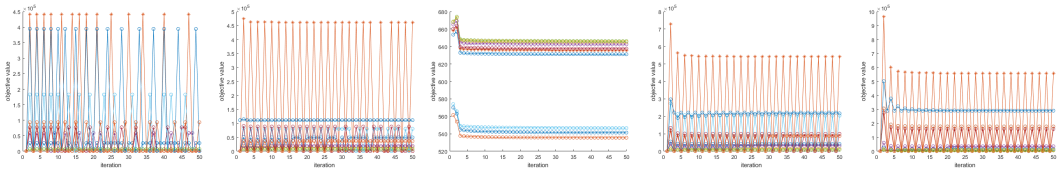
classification problems. The convergence behavior of the objective function under different parameter settings shows that the performance of the method can be influenced by the choice of parameters, such as C and λ.



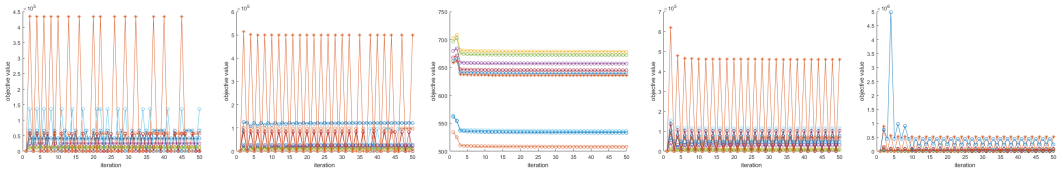
(a) C=0.01,$\lambda = 0.1$ (b) C=0.01,$\lambda = 0.3$ (c) C=0.01,$\lambda = 0.5$ (d) C=0.01,$\lambda = 0.7$ (e) C=0.01,$\lambda = 0.9$



(f) C=0.1,$\lambda = 0.1$ (g) C=0.1,$\lambda = 0.3$ (h) C=0.1,$\lambda = 0.5$ (i) C=0.1,$\lambda = 0.7$ (j) C=0.1,$\lambda = 0.9$



(k) C=1,$\lambda = 0.1$ (l) C=1,$\lambda = 0.3$ (m) C=1,$\lambda = 0.5$ (n) C=1,$\lambda = 0.7$ (o) C=1,$\lambda = 0.9$



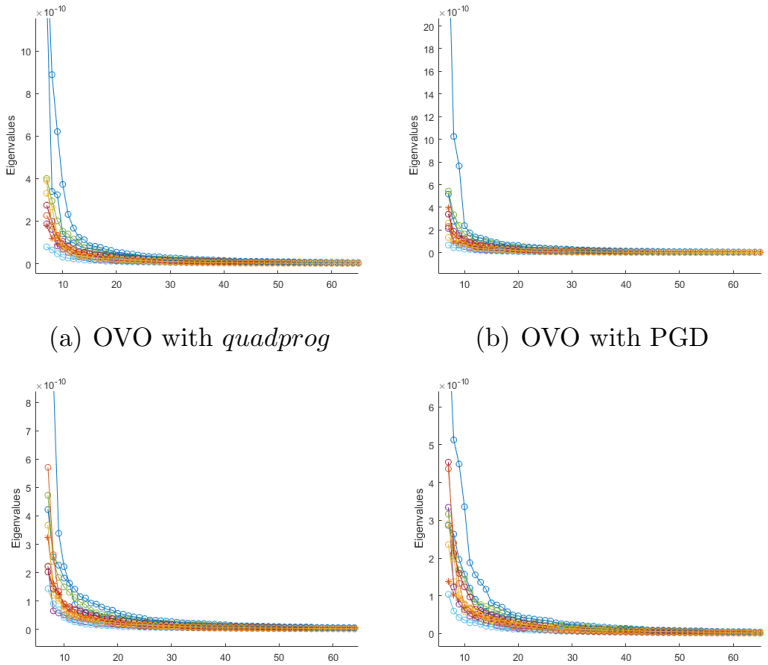
(p) C=10,$\lambda = 0.1$ (q) C=10,$\lambda = 0.3$ (r) C=10,$\lambda = 0.5$ (s) C=10,$\lambda = 0.7$ (t) C=10,$\lambda = 0.9$

Figure 5.4: The plots of OVO with PGD Method on **mfeat** Data Set

In particular, the results show that when $\lambda = 0.5$, the function value variation is the smallest, indicating that the choice of λ has a significant impact on the convergence of the algorithm. Additionally, the results of the alternate iterative experiments for solving $\boldsymbol{\alpha}$ using the projected gradient descent method show that when

$C = 1$ and $\lambda = 0.5$, the objective function values can decrease smoothly and reach the minimum value.

Moreover, the consistency of the results obtained by the quadprog solver and projected gradient descent method in both the OVR and OVO methods further validates the effectiveness of the proposed multi-view R-OPLS method in solving multi-classification problems. Additionally, the consistency of the results in the OVR and OVO methods suggests that the proposed method is not sensitive to the choice of multi-classification method. Overall, the results demonstrate that the proposed multi-view R-OPLS method, combined with appropriate parameter settings and optimization algorithms, can effectively solve multi-classification problems in multi-view data sets.



(a) OVO with *quadprog* (b) OVO with PGD

(c) OVR with *quadprog* (d) OVR with PGD

Figure 5.5: Eigenvalue plots on **mfeat** Data

When there are too few or too many features, these problems can affect the model's performance. Figure 5.5 (a) and (b) display the eigenvalue plots generated by the OVR approach, while Figure 5.5 (c) and (d) show the eigenvalue plots produced by the OVO method. The slope of the curve sharply decreases at some points, which are often called the "elbow point." Choosing the number of features to the left of the elbow point can explain most of the variance and avoid overfitting. However, if the goal is to maximize prediction accuracy, more features may be needed. Therefore, by observing the explained variance ratio curve, an appropriate number of features can be chosen to balance prediction accuracy. It's important to note that the OVO approach requires building $c(c-1)/2$ models, resulting in a total of 45 models for the **mfeat** data set. Therefore, for the convenience of viewing, only the eigenvalue change curves for the first ten models are presented in each plot of Figure 5.5 (c) and (d).

The following tables represent the results of the multi-view R-OPLS model, evaluated on different training sizes, 80%, 60%, 50%, 40%, and 20%. The evaluations are based on the accuracy of the models in predicting the output, with k representing the dimension of the output vector. Each table is split into two, one for each of the two different solvers, *quadprog* and PGD.

The table 5.2 presents classification accuracy results obtained using OVR method. As previously discussed, our innovative multi-view R-OPLS model has the capability to generate two classifiers at the same time using both R-OPLS and SVM techniques. This allows for a more comprehensive analysis of the data as it utilizes multiple perspectives to generate predictions. By combining these two approaches, our model can improve the accuracy and robustness of the classification results. Both R-OPLS and SVM achieve high classification accuracy, with R-OPLS generally performing slightly better in OVR method. For example, when $k = 20$ and

the training set size is 80%, R-OPLS achieves classification accuracy of 97.81% for *quadprog*, and when $k = 10$ and the training set size is 80%, R-OPLS achieves classification accuracy of 98.11% for PGD, while SVM achieves classification accuracy of 97.75% and 97.96%, respectively.

k	**Model**	**Training Size** 80%	60%	50%	40%	20%
(*quadprog*)						
$k = 30$	R-OPLS	$97.78 \pm 0.47\%$	$97.37 \pm 1.26\%$	$97.54 \pm 0.66\%$	$97.31 \pm 0.39\%$	$96.50 \pm 0.88\%$
	SVM	$97.67 \pm 1.07\%$	$97.23 \pm 1.40\%$	$97.29 \pm 0.91\%$	$97.03 \pm 0.36\%$	$96.21 \pm 0.60\%$
$k = 20$	R-OPLS	$97.81 \pm 1.19\%$	$97.83 \pm 1.42\%$	$97.66 \pm 0.94\%$	$97.44 \pm 0.48\%$	$96.70 \pm 0.86\%$
	SVM	$97.75 \pm 1.25\%$	$97.68 \pm 1.07\%$	$97.45 \pm 0.95\%$	$97.36 \pm 0.23\%$	$96.05 \pm 0.95\%$
$k = 10$	R-OPLS	$97.58 \pm 0.92\%$	$97.50 \pm 0.75\%$	$97.67 \pm 0.53\%$	$97.46 \pm 0.71\%$	$96.39 \pm 1.05\%$
	SVM	$97.62 \pm 0.63\%$	$97.43 \pm 0.45\%$	$97.46 \pm 0.34\%$	$97.15 \pm 0.44\%$	$95.98 \pm 0.57\%$
(PGD)						
$k = 30$	R-OPLS	$97.71 \pm 0.54\%$	$97.90 \pm 0.35\%$	$97.60 \pm 0.80\%$	$97.53 \pm 0.72\%$	$96.88 \pm 0.37\%$
	SVM	$97.71 \pm 0.29\%$	$97.92 \pm 0.58\%$	$97.66 \pm 0.54\%$	$97.54 \pm 0.63\%$	$96.91 \pm 0.47\%$
$k = 20$	R-OPLS	$97.91 \pm 0.84\%$	$97.78 \pm 0.85\%$	$97.44 \pm 0.56\%$	$97.60 \pm 0.73\%$	$96.64 \pm 0.55\%$
	SVM	$97.93 \pm 0.57\%$	$97.79 \pm 0.71\%$	$97.63 \pm 0.37\%$	$97.67 \pm 0.58\%$	$96.69 \pm 0.56\%$
$k = 10$	R-OPLS	$98.11 \pm 1.39\%$	$97.45 \pm 0.93\%$	$97.43 \pm 0.67\%$	$97.50 \pm 0.75\%$	$96.67 \pm 0.52\%$
	SVM	$97.96 \pm 1.54\%$	$97.39 \pm 1.11\%$	$97.49 \pm 0.61\%$	$97.50 \pm 0.42\%$	$96.66 \pm 0.40\%$

Table 5.2: OVR method for **mfeat** Data

Furthermore, the table 5.3 displays the classification accuracy outcomes for multi-view R-OPLS with the OVO technique utilizing both quadratic programming (quadprog) and projected gradient descent (PGD), and three different values of the parameter k (10, 20, and 30). Additionally, the table demonstrates the accuracy results for five distinct training sizes, varying from 20% to 80% of the total data set size. By using the OVO method, the results suggest that the SVM model outperforms the R-OPLS model in most cases, especially for the PGD solver. The SVM achieves

the highest accuracy of 93.92% for $k = 30$ and a training size of 80% with *quadprog*, and achieves the highest accuracy of 98.68% for $k = 30$ and a training size of 80% with PGD.

k	**Model**	**Training Size** 80%	60%	50%	40%	20%
(*quadprog*)						
$k = 30$	R-OPLS	$91.92 \pm 2.34\%$	$93.58 \pm 1.18\%$	$93.78 \pm 1.92\%$	$93.73 \pm 0.86\%$	$93.11 \pm 2.08\%$
	SVM	$93.92 \pm 1.59\%$	$91.35 \pm 2.41\%$	$89.69 \pm 1.42\%$	$87.64 \pm 2.04\%$	$76.98 \pm 3.34\%$
$k = 20$	R-OPLS	$92.92 \pm 2.34\%$	$93.37 \pm 2.01\%$	$93.52 \pm 1.09\%$	$93.71 \pm 0.88\%$	$93.10 \pm 2.03\%$
	SVM	$92.72 \pm 1.79\%$	$90.44 \pm 1.45\%$	$90.00 \pm 2.01\%$	$87.33 \pm 3.51\%$	$75.85 \pm 3.60\%$
$k = 10$	R-OPLS	$92.20 \pm 1.32\%$	$92.89 \pm 1.87\%$	$93.84 \pm 1.66\%$	$93.11 \pm 1.56\%$	$92.87 \pm 1.21\%$
	SVM	$93.17 \pm 1.34\%$	$90.53 \pm 1.11\%$	$89.02 \pm 2.09\%$	$87.55 \pm 2.29\%$	$76.06 \pm 1.45\%$
(PGD)						
$k = 30$	R-OPLS	$91.22 \pm 2.79\%$	$91.87 \pm 1.26\%$	$91.55 \pm 1.76\%$	$92.06 \pm 1.45\%$	$90.81 \pm 1.81\%$
	SVM	$98.68 \pm 1.07\%$	$98.57 \pm 0.56\%$	$98.41 \pm 0.49\%$	$98.17 \pm 0.33\%$	$96.82 \pm 0.62\%$
$k = 20$	R-OPLS	$90.92 \pm 2.85\%$	$91.24 \pm 1.27\%$	$91.83 \pm 1.38\%$	$91.72 \pm 1.79\%$	$90.47 \pm 1.47\%$
	SVM	$98.55 \pm 0.95\%$	$98.53 \pm 0.35\%$	$98.53 \pm 0.47\%$	$98.09 \pm 0.41\%$	$96.62 \pm 0.32\%$
$k = 10$	R-OPLS	$90.90 \pm 1.87\%$	$91.55 \pm 2.46\%$	$91.33 \pm 0.88\%$	$91.35 \pm 0.66\%$	$89.93 \pm 2.38\%$
	SVM	$98.63 \pm 0.62\%$	$98.15 \pm 0.60\%$	$98.34 \pm 0.36\%$	$98.20 \pm 0.55\%$	$96.73 \pm 0.71\%$

Table 5.3: OVO method for **mfeat** Data

In general, we can observe that as the training set size decreases, the classification accuracy also decreases. This is expected since smaller training sets provide less information to the model, making it more difficult to accurately classify new instances. However, the decrease in classification accuracy is relatively small, indicating that the models are robust and can still perform well even with smaller training sets.

The multi-view R-OPLS model has demonstrated remarkable performance in generating both OPLS- and SVM-based classifiers using different methods. The

OVR method showed comparable results for both the *quadprog* and PGD solvers. In contrast, when utilizing the OVO method, the experimental results of the SVM classifier in PGD were significantly improved, highlighting the strengths of the multi-view R-OPLS model. Overall, Multi-view R-OPLS can provide a classifier with better experimental results, irrespective of the method used.

CHAPTER 6

Conclusion

This thesis presents a unified model framework for multi-view learning that offers a comprehensive understanding of many existing methods from a regularized least squares perspective, and also inspires the development of new methods. In order to achieve high classification accuracy, our model projects the data into a low-dimensional subspace before performing classification. This approach ensures that important feature information is preserved while also improving the efficiency of the classification experiments. By projecting the data into a subspace with the lowest possible dimension, we can significantly reduce the time required for classification experiments, which is of great importance in practical applications. In addition, this approach also helps to simplify the complexity of the data, making it easier to visualize and interpret the results. Overall, the use of low-dimensional subspace projection is a powerful technique for achieving both accuracy and efficiency in classification tasks, which is widely used in various fields such as machine learning, data mining, and computer vision. Notably, the R-OPLS model can simultaneously generate two classifiers belonging to OPLS and SVM. The OPLS classifier focuses on extracting relevant features from the data and simplifying its structure, while the SVM classifier creates a decision boundary that maximizes the margin between different classes. This can greatly expand the range of uses, data types, and research domains to which our models are compatible. We validate our proposed through extensive experiments on binary and multi-class classification tasks in both single and multi-view settings. Our framework provides desirable flexibility for designing effective models across a

wide range of learning tasks. For instance, PCA, LDA, and sparse CCA [86, 82] can all be redefined under our framework, and can be extended to more than two views and non-linear representations. Moreover, our framework can be readily extended to other learning paradigms, such as semi-supervised multi-view learning and deep learning. Overall, this thesis's framework provides a strong foundation for future research in multi-view learning, enabling the development of more effective models across a wide range of learning tasks.

References

[1] *Mathworks help center*. `https://www.mathworks.com-help-index.html`, 2017.

[2] H. Abdi, *The method of least squares*, Encyclopedia of Measurement and Statistics. Thousand Oaks (CA): Sage, 2007.

[3] R. Ahuja, T. Magnanti, and J. Orlin, *Network flows*, Optimization Handbooks in Operations Research and Management Science, 1 (1988).

[4] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, *Random forests and decision trees*, UCSI International Journal of Computer Science, 9 (2012).

[5] G. Andrew, R. Arora, J. Bilmes, and K. Livescu, *Deep canonical correlation analysis*, International Conference on Machine Learning, PMLR, 28.

[6] J. Arenas-Garcia and G. Camps-Valls, *Efficient kernel orthonormalized pls for remote sensing applications*, IEEE Transactions on Geoscience and Remote Sensing, 46 (2008), pp. 2872–2881.

[7] S. Axler, *Linear Algebra Done Right*, Springer, 3rd ed., 2010.

[8] E. Birgin, J. Martinez, and M. Raydan, *Spectral projected gradient methods: Review and perspectives*, journal of statistical software, 60 (2014), pp. 1–21.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[10] A. Bjorck, *Least squares methods*, Handbook of Numerical Analysis, 1 (1990), pp. 465–652.

[11] L. Bottou, *Stochastic gradient descent tricks*, part of the Lecture Notes in Computer Science, 7700 (2012), pp. 421–436.

[12] N. Boumal, B. Mishra, P. Absil, and R. Sepulchre, *Manopt, a matlab toolbox for optimization on manifolds*, Journal of Machine Learning Research, 15 (2014), pp. 1455–1459.

[13] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge university press, Boston, MA, 2nd ed., 2004.

[14] L. Breiman, *Random forests*, Machine Learning, 45 (2001), pp. 5–32.

[15] G. Cao, A. Iosidis, K. Chen, and M. Gabbouj, *Generalized multi-view embedding for visual recognition and cross-modal retrieval*, IEEE Transactions on Cybernetics, 48 (2017), pp. 2542–2555.

[16] C. C. Chang and C. J. Lin, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, 2 (2011), pp. 27:1–27:27.

[17] J. Chem, *Random forest: A classification and regression tool for compound classification and qsar modeling*, Inf. Comput. Sci., 43 (2003), pp. 1947–1958.

[18] Y. Chien, *Pattern classification and scene analysis*, IEEE Transactions on Automatic Control, 19 (1974), pp. 462–463.

[19] L. Christian, *An overview of orthogonal partial least squares*, Towards Data Science, (2019).

[20] L. Drummond and A.N.Iusem, *A projected gradient method for vector optimization problems*, Computational Optimization and Applications, 28 (2004), pp. 5–29.

[21] D. Dua and C. Graff, *Uci machine learning repository.* `http://archive.ics.uci.edu-ml`, 2017.

[22] D. Faddeev and V. Faddeeva, *Computational mothods for linear algebra*, Freeman and Company, San Fransisco, 1956.

[23] T. Fearn, *On orthogonal signal correction*, Chemometr Intell Lab Syst., 50 (2000), pp. 47–52.

[24] G. Forsythe, *Pitfalls in computation, or why a math book isn't enough*, The American Mathematical Monthly, 77 (1970), pp. 931–956.

[25] P. Geladi and B. Kowalski, *Partial least-squares regression: A tutorial*, Analytica Chimica Acta, 185 (1986), pp. 1–7.

[26] B. Ghojogh, F. Karray, and M. Crowley, *Eigenvalue and generalized eigenvalue problems: Tutorial*, 2022, https://arxiv.org/abs/1903.11240.

[27] G. H. Golub and C. F. Van Loan, *Matrix computations*, The Johns Hopkins University Press, Baltimore and London, 3rd ed., 2012.

[28] A. Greenbaum, *Iterative methods for solving linear systems*, SIAM, Philadephia, 1997.

[29] P. Hansen, *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*, SIAM, Philadephia, 1998.

[30] H. Harold, *Relations between two sets of variates*, Biometrika, 28 (1936), pp. 321–377.

[31] T. Hastie, A. Buja, and R. Tibshirani, *Penalized discriminant analysis*, The Annals of Statistics, 23 (1995), pp. 73–102.

[32] T. Hastie and R. Tibshirani, *Discriminant adaptive nearest neighbor classification*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 18 (1996), pp. 607–616.

[33] H. Hill, *Handbook of Analytical Separations*, Elsevier, 4th ed., 2003.

[34] J. Hull, *A database for handwritten text recognition research*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (1994), pp. 550–554.

[35] M. Kan, S. Shan, H. Zhang, S. Lao, and X. Chen, *Multi-view discriminant analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2015), pp. 188–194.

[36] J. Keller, M. Gray, and J. Givens, *A fuzzy k-nearest neighbor algorithm*, IEEE Transactions on Systems, Man, and Cybernetics, 15 (1985), pp. 580–585.

[37] J. R. Kettenring, *Canonical analysis of several sets of variables*, Biometrika, 58 (1971), pp. 433–451.

[38] R. King, *A worldwide machine learning lab-openml.* `https://www.openml.org/search?type=data\&sort=runs\&id=40670\&status=active`, 2017.

[39] K. Kiran and R. Ramasubba, *An orthonormalized partial least squares based spatial filter for ssvep extraction*, the 2018 6International Conference on Intelligent Human Computer Interaction, 2008.

[40] O. Kramer, *Dimensionality Reduction with Unsupervised Nearest Neighbors*, Springer, Berlin, Heidelberg, 2013.

[41] P. Lai and C. Fyfe, *Kernel and nonlinear canonical correlation analysis*, International Journal of Neural Systems, 10 (2000), pp. 365–377.

[42] G. Lee and K. Lee, *Feature selection using distributions of orthogonal pls regression vectors in spectral data*, BioData Mining, 14 (2021).

[43] Z. Lei and S. Li, *Coupled spectral regression for matching heterogeneous faces*, 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 1123–1128. IEEE.

[44] G. Lekhana, *One hot encoding in machine learning.* `https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/`, 2017.

[45] D. Li, N. Dimitrova, M. Li, and I. K. Sethi, *Multimedia content processing through cross-modal association*, The Eleventh ACM International Conference on Multi-media, 2003, pp. 604–611.

[46] Y. Li, M. Yang, and Z. Zhang, *A survey of multi-view representation learning*, IEEE Transactions on Knowledge and Data Engineering, 31 (2015), pp. 1863–1883.

[47] A. Liaw and M. Wiener, *Classification and regression by randomforest*, R News, 3 (2002), pp. 1–41.

[48] A. Maida, *Cognitive computing: Theory and applications*, Handbook of Statistics, 35 (2016), pp. 2–384.

[49] S. Mawjoud, *Path loss propagation model prediction for gsm network planning*, International Journal of Computer Applications, (2013), pp. 30–33.

[50] S. Mehrkanoon, T. Falck, and J. A. K. Suykens, *Approximate solutions to ordinary differential equations using least squares support vector machines*, IEEE Transactions on Neural Networks and Learning Systems, 23 (2012), pp. 1356–1367.

[51] S. Mehrkanoon and J. Suykens, *Regularized semipaired kernel cca for domain adaptation*, IEEE tran. on Neural Networks and Learning System, 29 (2018), pp. 3199–3213.

[52] C. Micchelli and M. Pontil, *Learning the kernel function via regularization*, Journal of Machine Learning Research, 6 (2005), pp. 1099–1125.

[53] M.Pal, *Random forest classifier for remote sensing classification*, International Journal of Remote Sensing, 26 (2005).

[54] J. Nasiri, N. Charkari, and S. Jalili, *Least squares twin multi-class classification support vector machine*, Pattern Recognition, 48 (2015), pp. 984–992.

[55] K. Ng, *A Simple Explanation of Partial Least Square*, PhD thesis, Australian National University, 2013.

[56] A. Nielsen, *Multiset canonical correlations analysis and multispectral, truly multitemporal remote sensing data*, IEEE Transactions on Image Processing, 11 (2002), pp. 293–305.

[57] Z. Noumir, P. Honeine, and C. Richard, *Multi-class least squares classification at binaryclassification complexity*, Nice, France, 2011, IEEE workshop on Statistical Signal Processing (SSP), pp. 277–280.

[58] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Society for Industrial and Applied Mathematics, 1998.

[59] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*, Technical University of Denmark, 2012.

[60] B. T. Polyak, *Introduction to optimization*, Optimization Software, New York, 1987.

[61] N. Qian, *On the momentum term in gradient descent learning algorithms*, Neural Networks, 12 (1995), pp. 145–151.

[62] J. Randall and A. Rayner, *The accuracy of least squares calculations with the cholesky algorithm*, ELSEVIER, 127 (1990), pp. 463–502.

[63] S. Roweis and C. Brody, *Linear heteroencoders*, PhD thesis, University College London.

[64] S. Ruder, *An overview of gradient descent optimization algorithms.* `http://sebastianruder.com/optimizing-gradient-descent/index.html`, 2016.

[65] S. V. Salai, *gscatter3.* `https://www.mathworks.com/matlabcentral/fileexchange/37970-gscatter3`, 2022.

[66] R. Salakhutdinov and G. E. Hinton, *Efficient learning of deep boltzmann machines*, The Thirteenth International Conference on Artificial Intelligence and Statistics, 2010. PMLR.

[67] M. Sarkar and T. Y. Leong, *Application of k-nearest neighbors algorithm on breast cancer diagnosis problem*, Proc AMIA Symp., (2000), pp. 759–763.

[68] A. Sharma, A. Kumar, H. Daume, and D. Jacobs, *Generalized multiview analysis: A discriminative latent space*, 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 2160–2167. IEEE.

[69] L. Sun, S. Ji, and J. Ye, *Canonical correlation analysis for multilabel classification: A least squares formulation, extensions, and analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 33 (2010), pp. 194–200.

[70] S. Sun, *A survey of multi-view machine learning*, Neural Comput and Applic, 23 (2013), pp. 2031–2038.

[71] S. Sun and R. Huang, *An adaptive k-nearest neighbor algorithm*, Yantai, China, 2010, Seventh International Conference on Fuzzy Systems and Knowledge Discovery, pp. 91–94.

[72] S. Sun, X. Xie, and M. Yang, *Multiview uncorrelated discriminant analysis*, IEEE Transactions on Cybernetics, 46 (2015), pp. 3272–3284.

[73] D. Tomar and S. Agarwal, *A comparison on multi-class classification methods based on least squares twin support vector machine*, Knowledge-Based Systems, 81 (2015), pp. 131–147.

[74] M. Turk and A. Pentland, *Eigenfaces for recognition*, Journal of Cognitive Neuroscience, 3 (1991), pp. 71–86.

[75] V. Uurtio, J. Monteiro, J. Kandola, J. Shawe-Taylor, D. Fernandez-Reyes, and J. Rousu, *A tutorial on cannonical correlation methods*, ACM Computing Surveys, 50 (2017), pp. 1–33.

[76] J. Via, I. Santamaria, and J. Perez, *A learning algorithm for adaptive canonical correlation analysis of several data set*, Neural Networks, 20 (2007), pp. 137–152.

[77] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, *Extracting and composing robust features with denoising autoencoders*, the 25th international conference on Machine learning, 2018, pp. 1096–1103.

[78] H. Wang, S. Yan, D. Xu, X. Tang, and T. Huang, *Trace ratio vs. ratio trace for dimensionality reduction*, 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8. IEEE.

[79] L. Wang, R. Li, and W. Li, *Multiview orthonormalized partial least squares: Regularizations and deep extensions*, IEEE Transactions on Neural Networks and Learning Systems, (2021), pp. 1–15.

[80] L. Wang, L. Zhang, C. Shen, , and R. Li, *Uncorrelated semi-paired subspace learning*, (2020).

[81] J. H. Wilkinson, *The algebraic eigenvalue problem*, Springer, Oxford Clarendon, 1st ed., 1965.

[82] D. Witten and R. Tibshirani, *Extensions of sparse canonical correlation analysis with applications to genomic data*, Statistical Applications in Genetics and Molecular Biology, 8 (2009), pp. 159–194.

[83] S. Wold, C. Albano, W. Dunn, U. Edlund, K. Esbensen, P. Geladi, S. Hellberg, E. Johansson, W. Lindberg, and M. Sjostrom, *Multivariate data analysis in chemistry*, Chemometrics, (1984), pp. 17–95.

[84] K. Worsley, J. Poline, K. Friston, and A. Evans, *Characterizing the response of pet and fmri data using multivariate linear models*, Neuroimage, 6 (1997), pp. 305–319.

[85] S. Xiang, F. Nie, G. Meng, C. Pan, and C. Zhang, *Discriminative least squares regression for multiclass classification and feature selection*, IEEE Transactions on Neural Networks and Learning Systems, 23 (2012), pp. 1738–1754.

[86] M. Xu, Z. Zhu, X. Zhang, Y. Zhao, and X. Li, *Canonical correlation analysis with l2,1-norm for multiview data representation*, IEEE Transactions on Cybernetics, 50 (2020), pp. 4772–4782.

[87] J. Ye, *Least squares linear discriminant analysis*, Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 1087–1093.

[88] J. Ye and T. Xiong, *Svm versus least squares svm*, the Eleventh International Conference on Artificial Intelligence and Statistics, 2007, pp. 644–651. PMLR.

[89] L. Zhang, L. Wang, Z. Bai, and R. Li, *A self-consistent-field iteration for orthogonal canonical correlation analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 44 (2022), pp. 890–904.

[90] N. Zhang, D. Lei, and J. Zhao, *An improved adagrad gradient descent optimization algorithm*, 2018 Chinese Automation Congress (CAC), 2018, pp. 2359—-2362.

Biographical Statement

Ce Bian was born in Xuanhua, HEBEI, China in 1987. He received his B.S. degree from Agricultural University of Hebei, China, in 2009. His M.S. degree from The University of Texas at Arlington in Mathematics department at 2016. Ce Bian started his Ph.D. program at Fall 2017, and jointed the data science research group since Spring 2019.